
الدكتور عبد الحسن الحسيني
الخوارزميات
والبرمجة الانشائية بلغة
باسكال
Bibliotheca Alexandrina
0007956



الخوارزميات
والبرمجة الانشائية بلغة
باسكال

الطبعة الأولى

1408 هـ - 1988 م

المؤسسة الجامعية للدراسات والنشر والتوزيع

بيروت - الحمراء - شارع اميل اده - بناية سلام
هاتف ٨٠٢٤٢٨ - ٨٠٢٤٠٧ - ٨٠٢٢٩٦
بيروت - المصيطبة - بناية طاهر هاتف : ٣٠١٠٣٠ - ٣١١٣١٠
ص . ب: ٦٣١١ / ١١٣ تلكس: LE ٢٠٦٦٥ - ٢٠٦٨٠ لبنان

سلسلة بإشراف
د. عبد الحسن الحسيني

الخوارزميات والبرمجة الانشائية بلغة باسكال

المؤسسة الجامعية للدراسات والنشر والتوزيع



مقدمة

مع تطوُّر صناعة الآلات الحاسبة الإلكترونية « الكومبيوتر » ، وظهور نماذج مختلفة الأحجام والمقدرات ، بدأ الصانعون يفكرون بتطوير لغات البرمجة وطرقها، فكان إن ظهرت لغات أكثر أو أقل تعقيداً موجهة نحو علوم مُتخصِّصة ، أو اللغات المتخصصة . مثلاً: ألغول (ALGOL) هي من اللغات المتخصصة بحلّ المسائل الرياضية والخوارزميات ، فورتران (FORTRAN) هي لغة مُتخصصة بحلّ المسائل العلمية والرياضية، كوبول وهي لغة متخصصة بحلّ المسائل الاقتصادية والإدارية؛ LISP لغة متخصصة بمعالجة النصوص؛ GPSS (general purpose simulation system) وهي لغة متخصصة بحلّ المسائل ذات الطابع العشوائي أو المسائل التي تعتمد على الحالات الطارئة . . . ولقد حاول الصانعون وشركات البرامج أو ما يسمى ببيوت المناهج (softow are house)توسيع لغاتهم أو إنشاء لغات جديدةتستطيع العمل في أكثر من مجال واحد فكان أن طوروا لغة بازيك وفورتران،وظهرت لغة PL/1 كلغة متخصصة بالعلوم والإدارة ، ومن ثم ظهرت لغات متعددة الاستعمال كلغة باسكال (PASCAL) وآدا (ADA) ، هذه اللغات تستطيع الإجابة على أكثر حاجات المستعملين العلمية والإدارية ، ولكن لكل منها ، مساوىء وحسنات بالمقارنة مع اللغات المتخصصة في إطار الاختصاص الذي تستعمل به هذه اللغة .

كما ظهرت برامج ورُزم برامج موجهة نحو عملٍ معين ، أو باتجاه تطبيق معين .

هكذا ، فعملية البرمجة هي العملية الأكثر تعقيداً وصعوبة والتي تواجه المُستعمِل أو المعلوماتي ، وهي تتألف من مرحلتين أساسيتين : مرحلة التحليل ، أي تحليل المسألة (Analyse) ، ومرحلة التكويد أو صياغة الحلّ بواسطة لغة خاصة بالبرمجة . لذا فقد حاول

العاملون في حقل المعلوماتية تبسيط هذه العملية لزيادة سرعة وفعالية المبرمجين والُمحلّلين ، وجعل تحاليلهم صالحة للعمل على أكثر من مكنة وقابلة للبرمجة أو التكويد بواسطة عدة لغات مختلفة .

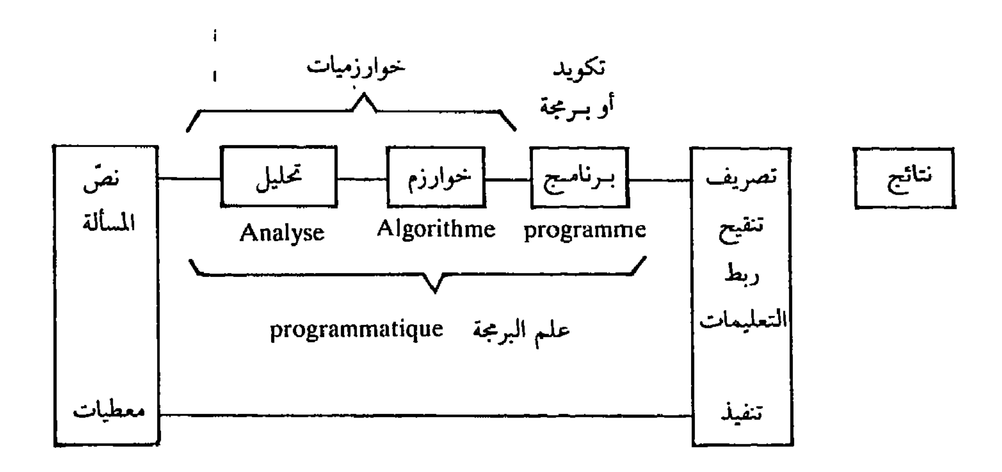
من هنا ظهرت البرمجة المتعددة والبرمجة بالتوازي ،ومن ثم بُدىء باستعمال البرمجة الانشائية التي تقوم على تبسيط المسألة بواسطة تجزئة الحلّ إلى أقسام عديدة ، مما يجعل عملية اختبار صحة عمل البرنامج والتدقيق بالأخطاء اللغوية والمنطقية أكثر سهولةٍ ويتم بسرعة أكبر ، كما إن إمكانية تطوير البرنامج أو تعديله تصبح أكثر سهولة .

هكذا فهذا الكتاب يعالج البرمجة الانشائية من خلال الخوارزميات التي تؤدي إلى تبسيط عملية البرمجة للحصول على النتائج بعد تلقيمه بالمعطيات الحقيقية .

إن التعبير : خوارزم + معطيات = برنامج ، هو من التعابير الصحيحة والعملية الفعالة ، ويُعتبر طريقة فعالة للحصول على برنامج صحيح ، دون الخوف من الوقوع في الأخطاء اللغوية (Syntax error) أو المنطقية (Logic error) .

ولقد إعتمدنا في هذا الكتاب ، لغة باسكال (PASCAL) ، وهي من اللغات الأكثر شيوعاً في الجامعات اليوم ، والكثيرة الاستعمال لتكويد الخوارزميات ، كونها لغة مُتخصّصة بالبرمجة كما وتعتبر من اللغات ذات الاستعمال العام في مجالات علمية ، رياضية ، إقتصادية إدارية مختلفة .

وبالإمكان أن نوجز عملية حلّ المسألة بواسطة المخطط التالي :



المراحل الكبرى الأساسية لصياغة الخوارزم :

أ ـ المرحلة الأولى : تحضير المعالجة والتحليل

وتقوم على تبيان المعطيات الضرورية لحلّ المسألة . وتبيان ما هو موجود وما هو مطلوب ، ورسم المتحولات والسجلات وتحديد أبعادها وحجمها .

ب ـ المرحلة الثانية : المعالجة

حلّ المسألة خطوة بعد خطوة . بعد تقسيمها إلى مراحل ثانوية إذا كان ذلك ضرورياً ومفيداً في آن .

ج ـ المرحلة الثالثة : إخراج النتائج

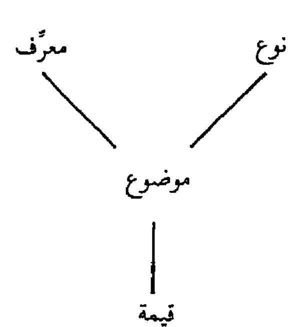
طباعة النتائج المطلوبة على شاشة أو على الورق .

تتم المعالجة بواسطة تعليمات خاصة تجري على مواضيع (ثوابت عددية ورمزية) أو على معلومات مختلفة . كل موضوع (object) أو معلومة يحتوي على ثلاثة خصائص :

ـ المعرِّف (identificator) : وهو إسم الموضوع المستعمل في الخوارزم وفي البرنامج . لكل موضوع معرِّف واحد .

ـ النوع (type) ، يُحدِّد المجموعة التي يأخذ الموضوع قيمته منها .

ـ القيمة (value) : وهي مجموعة مختلفة من ضمن المجموعة المحدَّدة في التعريف عن النوع .



الفصل الأول

الخوارزميات

1.1 ـ المسار المعلوماتي لحلّ المسألة

يظهر المسار المعلوماتي على الشكل التالي :

المسألة ←البرمجة— البرنامج + المعطيات ←التنفيذ— النتائج

تختلف المسائل التي قد يواجهها الانسان في نوعيتها وتعقيدها ، فالعامل في حقل الفضاء تواجهه مسائل لها طابع فيزيائي ، كهربائي ، رياضي ، هيدروليكي ، . . . الخ . هذه المسائل هي شديدة التعقيد كونها تتأثر بعوامل مختلفة غير منظورة أو ملموسة من قبل الانسان ، كما وتتأثر بالعوامل الطبيعية ، وبعوامل لها طابع « الصدفة » أو العشوائية في الحدوث . أما العامل في حقل الصناعة ، فتواجهه مشاكل لها طابع كهربائي فيزيائي ، ديناميكي ، رياضي تتأثر بالوقت الفعلي للزمن . . . (real time) وتعقيدها أقل من تلك التي تتأثر بالعوامل الغير منظورة أو الغير ملموسة . . أما العامل في حقل الادارة فتواجهه مشاكل ومسائل لها طابع إقتصادي ، تنظيمي ، حسابي . . . الخ ، هذه المسائل هي الأقل تعقيداً ، لأنها منظورة من قبل الانسان وتتأثر به ، ولكنها تتطلب عمليات إدخال وإخراج كثيرة ومعقدة وتحتاج إلى إمكانيات تخزين كبيرة . . . الخ .

ولقد إعتمد الصانعون والعاملون في المعلوماتية لغات مختلفة تتوافق مع طابع كل مسألة وتعقيدها ، وذلك بغرض تسهيل البرمجة والحصول على النتائج ، من هنا فلقد ظهرت لغات متخصصة بالعلوم الرياضية (فورتران ، الغول . . .) ولغات خاصة بالادارة (كوبول . . .) ، ولغات خاصة بمعالجة النصوص (LISP...) وتطورت هذه اللغات حتى أصبحت تغطي جزئياً أو كلياً الأعمال ذات الطابع المختلف ، كما تطورت معها طريقة البرمجة . . . كل ذلك من أجل تحسين العمل وتسريعه . ومع تطور الآلات الحاسبة « الكومبيوتر » وظهور أنواع كثيرة بأحجام مختلفة ، وفعالية متفاوتة ، وبإمكانيات

متقدمة ، وبأسعار مشجعة ، قام الصانعون بتقديم وعرض لغات مختلفة للبرمجة جديدة وتطوير القديم منها بحيث تتلاءم مع حاجات المستعملين لوسائل المعلوماتية فكان أن تطورت اللغات فورتران ، كوبول ، ألغول ، وظهرت أخرى باسكال ، آدا ، LISP ، C . . . ومع اللغات بالصيغ الجديدة عرض الصانعون برامج التصريف المختلفة (Compiler) أو المصرِّفات التي تقوم بترجمة البرنامج من اللغات المتطورة ذات المستوى العالي (فورتران ، كوبول ، باسكال . .) الى لغة الآلة أو اللغة القابلة للتنفيذ (object program) ، كما عرض الصانعون لتسريع البرمجة مفسِّرات (interpretator) ، خاصة تقوم بترجمة البرنامج إلى لغة الآلة ، وتنفيذه مباشرة وذلك خطوة خطوة أو تعليمة بعد تعليمة (instruction) عند إدخال كل منها بواسطة لوحة الملامس (Key bord) المرتبطة بالأداة الطرفية (terminal) والمتصلة بالحاسب أو الكومبيوتر .

إضافة لذلك ، فقد قامت شركات المعلوماتية بانتاج وعرض رُزم من البرامج جاهزة للتطبيق والاستعمال لحلّ مسألة محددة ودقيقة ، إضافة الى رزم من المناهج أو البرامج الجاهزة للاستعمال والتطبيق بعد تعديلها لتلائم حلّ المسألة المطروحة لدى المستعمل (progiciel) . هكذا ، فمهمة الإنسان تكمن في برمجة المسألة المطلوب حلّها ، أما مهمة المكنة فتنحصر بعملية تنفيذ البرنامج الموضوع من قِبل المُستعْمِل ، وهذا يتم في ثلاث مراحل أساسية : المرحلة الأولى ، وهي مرحلة إدخال البرنامج إلى المكنة وتنقيحه من الأخطاء اللغوية (syntax error) الموجودة فيه، والثانية ترجمته بواسطة المصرِّف (compiler) إلى لغة مفهومة من قِبل المكنة (Machine language) ومن ثم معالجته بواسطة مُنقِّح الأربطة (link editor) (في حال عدم إستعمال المفسِّر مباشرة لترجمة البرنامج) بالنسبة للبرامج المكتوبة باللغات ذات المستوى العالي .H.L.L (High level language) والمرحلة الأخيرة ، هي عبارة عن تنفيذ البرنامج بعد تلقيحه بالمعطيات الحقيقية للحصول على النتائج التي نبحث عنها .

وهناك عاملان يجعلان من البرمجة عملاً صعباً . من جهة ، هناك المسافة بين المسألة المطروحة والبرنامج ، ومن جهة أخرى ، الأشكال المختلفة التي يمكن أن يأخذها البرنامج .

العرض الأول للمسألة المطروحة أمام المعلوماتي (المحلِّل أو المبرمج) يكون مُصاغاً بلغة طبيعية (مثلاً باللغة العربية) ، مما يجعله عادة ضبابياً ، غير متماسكاً وغير كاملاً . هذا العرض يُحدِّد مميزات النتائج المطلوبة في حلّ المسألة .

وعلى العكس ، يُكتب البرنامج بلغة إصطناعية شكلية مُحدَّدة ، وهو يُحدِّد الطريقة الحسابية التي تسمح من خلال بعض المعطيات ، بالحصول على النتيجة المطلوبة . ولكي نعبر من عرض المسألة إلى صياغة البرنامج ، يجب إذن :

- حلّ المسألة : أي إيجاد مُخطط للحساب ، أو طريقة للحساب تؤدي للحصول على النتائج المطلوبة من خلال المعطيات المعروفة التي يقوم بإدخالها المبرمج أو المُحلِّل ، وهذه هي عملية التحليل والحلّ . هذه العملية تؤدي إلى صياغة الخوارزم (algorithme) أو السياق البياني (organigramme) .
- الإفصاح عن هذا الحل أو صياغته وتحديده بواسطة لغة مقبولة من النظام المعلوماتي الذي نعمل عليه ، وهذه هي عملية التكويد أو البرمجة (programatic) .

هكذا وبإيجاز : فإن المدخل إلى وضع الخوارزم أو المدخل إلى الخوارزميات يتم هنا بواسطة لغة للوصف سهلة وبسيطة من الناحية النحوية ، ولكنها عامة في مستوى الإنشاءات المسموحة . وعملية إتقان البرمجة تتم إذن ، بواسطة عملية تكويد بسيطة حسب مخططات الترجمة النموذجية .

هذا المفهوم يسمح بسهولة من العبور من لغة للبرمجة أي لغة أخرى ، لأننا نكون قد فصلنا عملية تحليل المسألة عن عملية تكويدها .

هكذا يُدعى خوارزم (algorithme) ، عملية وصف العمليات الضرورية التي وبواسطتها ، من خلال قيم للادخال تُدعى معطيات (data) ، نحصل على نتيجة أو نتائج مُحدَّدة .

البرنامج هو عبارة عن خوارزم مكتوب بلغة مُحدَّدة وخاصة ، وذلك باستعمال مصطلحات خاصة مؤلفة من رموز وسمات أساسية ، تُؤلف كلماتٍ وجملٍ مُشكَّلة بواسطة نحو (syntax) خاص ودقيق ، لتؤلِّف دلالة دقيقة دلالة (semantic) ، تُمثِّل أوامر وتعليمات تفهمها الآلة وتستطيع تنفيذها .

هكذا ، ولنفترض وجود مسألة معينة ، نستطيع تمييز مرحلتين أساسيتين هما :

البحث عن حلّ واضح ، يُدعى خوارزم (algorithme) أو تسلسل منطقي أو سياق بياني واضح . هذه المرحلة تدعى مرحلة تحليل المسألة .

التعبير عن الخوارزم بواسطة لغة للبرمجة ، بغرض إستعمال الحلّ بواسطة الكومبيوتر . هذه المرحلة الثانية تدعى تكويد الخوارزم (codefication) .

أما البرمجة الإنشائية فهي تعني صياغة الخوارزم وبالتالي البرامج بشكل يكون فيه هذا الأخير قابلاً للتطوير والتعديل دون المساس بجوهر البرنامج أو جوهر الخوارزم ، ويعتمد ذلك على أساس تجزئة المسألة وتجزئة الحلّ إلى مسائل وحلول ثانوية تُجمع مع بعضها لتؤلف وحدة متكاملة .

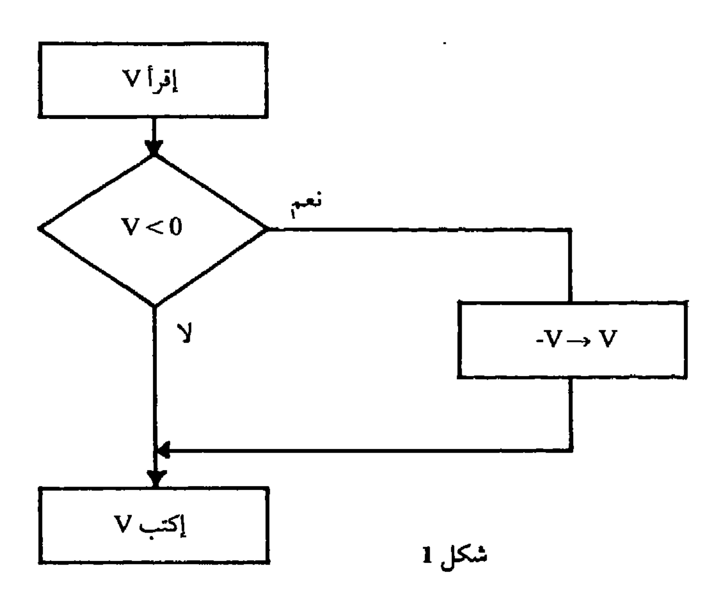
1.2 ـ تمثيل الخوارزم (algorithme representation)

الخوارزم هو عبارة عن مجموعة من الأوامر أو الأفعال . هذه المجموعة هي مُركّبة وإنشائية ، لأن ترتيب الأفعال والأوامر بداخل الخوارزم هو أساسي ويُحددها مسار سير المعلومات والنتائج الجزئية والنهائية .

لتمثيل هذه المجموعة الإنشائية من الأفعال ، سنقوم باختبار طريقتين للعمل :

ـ الطريقة الأولى وتقوم على وضع كل فصل بداخل « علبة » ، (أو بداخل مستطيل) ، بين هذه المستطيلات أو العلب يوجد أسهم تحدِّد نظام تسلسل وترتيب الأفعال والعمليات والعلاقات فيما بينها .

مثلاً :



شكل 1

تعرف هذه الطريقة بالإسم السياق البياني أو organigramme بالفرنسية ، كما وتدعى بالإسم Algorithme باللغة الإنكليزية .

ـ الطريقة الثانية تقوم على إستعمال سلسلة « بينات » و« نصـوص » «enoncés» (أي شروحات أو عرض مكتوب لكل عملية من الخوارزم) ، بعض هذه البينـات يُمثّـل عمليات بسيطة ، وأخرى تمثل عمليات أكثر تعقيداً وتسمح بتمثيل عملية ربط مختلف الأوامر والعمليات فيما بينها .

هكذا ، فالخوارزم الموجود في الشكل 1 ، يُمكن أن يُكتب بواسطة التعبير A :

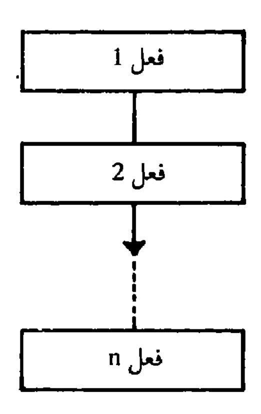
(A)

```
Var V : INTEGER;
READ V
IF V < 0       THEN V ←-V
ECRIRE V
```

أو بالعربية : تصريح V : صحيح
إقرأ V
إذا كان V أصغر من صفر إذن : V ←-V
إكتب V .

1.3 ـ ربط الأفعال على التوالي . البينات المركبة

في السياق البياني ، يكفي ربط المستطيلات التي تُمثّـل الأفعال بالطريقة التالية :



في التعبير الخوارزمي ، سنُشكِّـل بنية مُركّـبة بترتيب البينات الواحدة تلو الأخرى ، وذلك إما بتغيير السطر ، وإما باستعمال فاصل ، مثلاً نقطة فاصلة (؛) .

enoncé 1;	بنية 1 ؛
enoncé 2; enoncé 3;	بنية 2 ؛ بنية 3 ؛
⋮	⋮
enoncé n;	بنية n ؛

لتجميع بعض البينات في مجموعة ، من الملائم إستعمال الكلمات begin (debut) و(end (fin (بداية ونهاية) كأهلة أو كأدوات لحصر البينات .

Start start enoncé 1; enoncé 2 end enoncé 3; ... enoncé n end

بداية بداية بينة 1 ؛ بينة 2 نهاية بينة 3 ؛ . . . بينة n نهاية .

أو بشكل أفضل :

بداية

بداية بينة 1 ؛

بينة 2 ؛

نهاية

بينة 3 ؛

⋮

بينة n ؛

نهاية

1.4 ـ التخصيص (ASSIGNMENT, AFFECTATION)

الفعل البسيط هو التخصيص ، ويقوم على تخصيص قيمة معينة يمكن أن تكون قيمة تعبير جبري أو منطقي إلى متحولة من نفس النوع . لتمثيل فعل التخصيص سنستعمل الاشارة ← ونرمز إليه بالتعبير التالي :

$V \leftarrow E$ ، حيث V هو معرِّف (identificator) للمتحولة وE عبارة عن تعبير رياضي (جبري ، منطقي) من نفس نوعية المتحولة ؛ أي إذا كانت المتحولة يصرَّح عنها على أنها متحولة من نوع حقيقي فيجب أن تكون قيمة التعبير الرياضي حقيقية .

من الممكن إعتبار عمليات الإدخال ـ الإخراج كعمليات تخصيص خاصة :

الأمر « إقرأ V » يعني وجود مُعطى معين ، على جهاز محيطي للإدخال ، وهذا المعطى هو جاهز لكي يكون مقروءاً ، وقيمته سيتم تخصيصها أو منحها للمتحولة V

الأمر «Ecrire v» (إكتب v) ، يعني تخصيص قيمة المتحولة إلى جهاز الإخراج المحيطي للحاسب .

مثلاً : (نستعمل ثلاثة أجهزة لبسط الأشرطة المغناطيسية)

إقرأ X على BM1

إقرأ Y على BM2

$Z \leftarrow x + y$

إكتب Z على BM3

1.5 ـ التعاقب (Alternative)

1.5.1 ـ للتعاقب فرعين :

في مستوى السياق البياني ، يُوضع التعبير المنطقي المطلوب تقييمه في علبة أو مستطيل على شكل مُعين (Losange) بمخرجين ، يرتبط المخرج الأول للمعين بالشرط الصحيح (TRUE) ، والإخراج بالشرط الخطأ (FALSE) .

في التعبير الخوارزمي المكتوب ، سيكون معنا أحد النصوص الشرطية التالية :

IF condition B **THEN** STATEMENT 1 **ELSE** STATEMENT 2

IF condition B **THEN** statement

أو بالفرنسية

Si Condition B **alors** enoncé

أو بالعربية :

إذا الشرط B إذن العملية 1

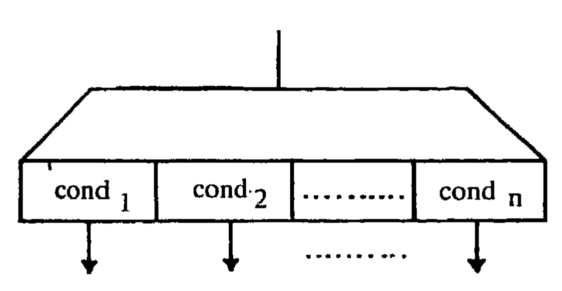
وإلا العملية 2 .

ترجمة هذه النصوص أو تكويدها في لغة باسكال ستبدو على الشكل التالي :

```
IF C THEN begin
          statement 1
          statement 2
          end;
      ELSE begin
           statement
           :
           end
```

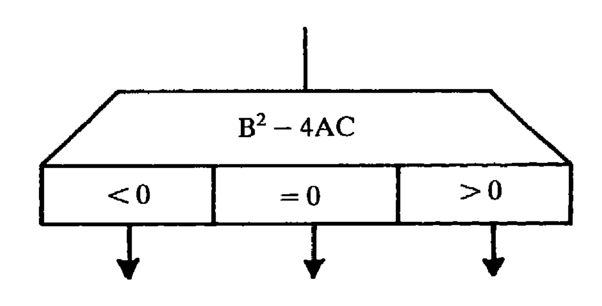
1.5.2 ـ التعاقب بعدة فروع

بدلاً من إعتماد حالتين مرتبطتين بالقيم « حقيقية » (TRUE) و« غلط » (FALSE) لتعبير بولي منطقي ، قد يكون مريحاً في بعض الأحيان أن نقوم بتمثيل عملية إختيار لحالة واحدة من بين عدد n من الإمكانيات (n ـ ثانية معروفة) . سنحصل إذن ، في السياق البياني ، على علبة تحتوي على عدد n من المخارج ، كلٍ منها يرتبط بالقيمة « حقيقة » لأحد الشروط .



Cond n ← عبارة عن الشرط رقم n .

مثلاً :



في التعبير الخوارزمي ، سنستعمل الأمر «in case of» (في الحالة حيث) أو بالفرنسية au cas où :

in case of	في الحالة حيث :
cond 1: statement 1	شروط 1 : بينة 1
cond 2 : statement 2	شرط 2 : بينة 2
...	. . . . .
cond n : statement n	شرط n : بينة n

(البينة n تعني العملية أو الفعل المطلوب تنفيذه عندما يكون الشرط n نافذاً أي بقيمة تعادل « حقيقة » .

مثلاً :

السيد سمير إشترى كمية معينة من منتوجة حيث ثمن الوحدة منها يساوي 50 ليرة . إذا كان الثمن المطلوب دفعه أقل من 200 ليرة ، يجب إضافة 25 ليرة بدل مصاريف نقل . المطلوب إخراج وكتابة مجموع المبلغ على الفاتورة الخاصة بالسيد سمير .

من الممكن كتابة الحلّ بالتعبير الخوارزمي التالي :

```
Var Q , PBRUT, PNET : REAL
Lire Q
PBRUT ← Q * 50
```

من الممكن كتابة الحلّ بالتعبير الخوارزمي التالي :

```
Var Q, PBRUT, PNET : REAL
     Read Q
PBRUT ← Q * 50
if PBRUT ≥ 200 then PNET ← PBRUT
                    else PNET ← PBRUT + 25
write PNET
```

PBRUT ـ الثمن الاجمالي
PNET ـ الثمن الصافي .

1.6 ـ التكرار (repetition)

من الضروري في بعض الأحيان أن نقوم بتنفيذ فعل معين أو مجموعة من الأفعال أو العمليات لعدد من المرتبات . وهذا ما يدعى بحلقة الحساب أو حلقة التكرار (loop) .

1.6.1 ـ الحلقات : while (طالما) والحلقة TO (حتى) .

يُرمز إلى هذه الحلقات على الشكل التالي :

```
While condition C do begin
    statements
    end
```

أي : طالما الشرط C إعمل بداية
أفعال
نهاية
أو :

```
repeat Begin
    statement
  UNTIL condition c;
```

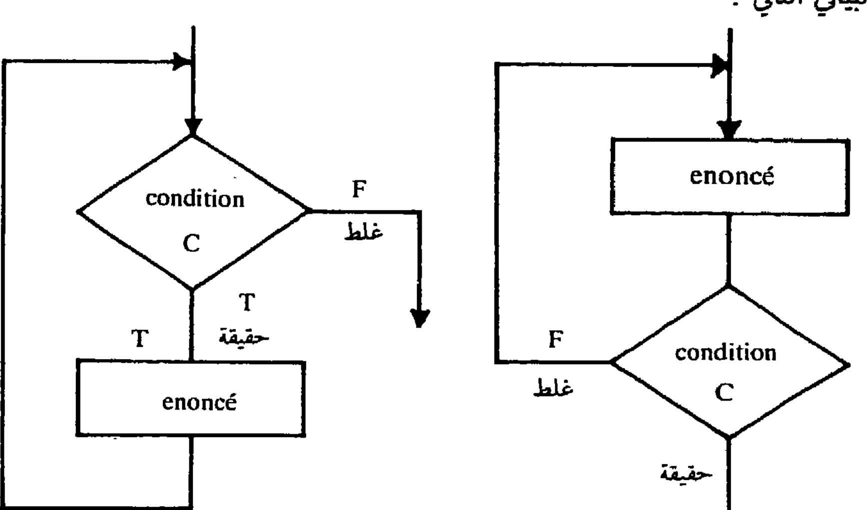
كرِّر الأفعال حتى الشرط C
أو :

```
While condition C do begin
        statements
        end
```

طالما الشرط C إعمل بداية
أفعال
نهاية

كرِّر الأفعال (التعليمات) حتى الشرط C

الحلقات tant que (طالما) و TO (jusqu'à) (حتى) تناسب على التوالي السياق البياني التالي :



```
While cond do begin
 statements
end.
```

1.6.2 ـ الحلقة For

يتعلق ذلك بعملية تكرار العمليات باستعمال عدّاد (counter) وذلك على الشكل التالي :

For I EQ M TON, step p, regeat actions

I ـ عبارة عن متحولة تتغير قيمتها من M إلى N وتُمثّل العدّاد .

P ـ الخطوة التي تتغير فيها القيمة I .

معنى هذا الأمر هو التالي :

إلى I من M إلى N ، خطوة P ، كرِّر الأفعال

هنا ، I ـ عبارة عن معرِّف يُمثّل متحولة صحيحة (عدّاد) ، M ، N ، P عبارة عن ثوابت صحيحة أو معرِّفات بمتحولات صحيحة أو تعابير جبرية بقيمة صحيحة .

مثلاً : إكتب الخوارزم الذي يحسب الدالة العاملية n أو F = n!

```
Var N, F, I : INTEGER;
Read N [ N ≥ 0 ] ;
    F ← I;
IF N > 0 THEN FOR I EQ1 TON REPEAT
    F ← F * I
WRITE F;
END
```

ترجمة الحلقة FOR بلغة باسكال تتم بواسطة الأشكال التالية :

```
1) Repeat
    i = i + 1
    :
    c = condition
UNTIL C
```

مثل 2 :

مجموع عناصر الجدول

Var S : REAL;	{ التصريح عن المجموع S : متحولة حقيقية }
Var I : integer;	{ التصريح عن I : عدد صحيح }
Var TAB (50) : REAL	{ التصريح عن الجدول TAB الذي يتألف من 50 عنصر من نوع حقيقي }
For I = 1 TO 50 repeat	{ كرِّر من I = 1 حتى 50 }
read A(I)	{ إقرأ A(I) }
S ← 0	
for I = 1 TO 50 repeat	{ من I = 1 حتى 50 كرِّر }
S = S + A (I)	{ إحسب المجموع S = S + A(I) }
write S	{ إكتب S }

ترجمة أو تكويد الحلقة For (Pour) أو « إلى » بلغة باسكال تبدو على الشكل التالي :

```
1) repeat
     i = i .01 1
     .
     .
     c = condition
Until C
2) for        i : = 1 TO n DO
               begin
              statements
              end;
3) While C    Begin
              .
              .
             end;

4) repeat
  statements
  until not C
```

الفصل الثاني

التحليل التصاعدي والانحداري

يلعب الخوارزم الخاص بحلّ مسألة معينة دور الوسيط بين المسألة بحد ذاتها والبرنامج المكتوب بلغة معينة . يُكتب هذا الخوارزم باستعمال تعابير خاصة تدعى تعابير خوارزمية (نسبةٍ إلى الحلّ المنصوص بلغة معينة كالفرنسية أو العربية أو غير ذلك) ، أو يُمثّل بواسطة سياق بياني (algorithme, organigramme) . ولكن المرحلة الأصعب في البرمجة ككل ، ليست عملية تكويد الخوارزم أو ترجمته إلى لغة معينة ، بل هي عملية تصوّر الخوارزم نفسه أي تصوّر طريقة الحلّ الأمثل ، وهذه هي مرحلة التحليل (analyse) .

لتسهيل العمل ولتحسينه ، سنعمل لفصل الصعوبات عن بعضها ، أو تقسيم المسألة الأولى إلى مسائل أو مواضيع بمستويات مختلفة من التعقيد . هذا هو المسار أو الطريق الواجب إتباعه لحلّ المسألة وصياغة الخوارزم . من الطرق الواجب اتباعها لبلوغ الطريقة المثلى للحلّ المطلوب ، نرى من المفيد الحديث ولو بإيجاز عن الطريقة التصاعدية والطريقة الانحدارية في التحليل .

2.1 ـ التحليل الانحداري

تقوم هذه الطريقة ، ومن خلال المسألة المطروحة في حالتها الأولى ، على تقسيم المسألة و« الانحدار » مرحلة بعد أخرى نحو مجموعات من المسائل الصغيرة السهلة الحلّ .

2.2 ـ التحليل التصاعدي

المسار الموضوع يقوم (على عكس التحليل الانحداري الذي يبدأ من المستوى الأعلى) ، ومن خلال المستوى الأدنى على الصعود خطوة بعد خطوة نحو المسألة المطلوب حلّها . نبدأ بحلّ المسائل الأسهل التي نعرف بأنها ستساعد على حلّ المسألة كاملةٍ بحالتها الأولى .

وبطريقة أخرى ، نقوم بصنع وسائط وأدوات خاصة ، وفي مرحلة داخلية نقـوم بتعريف إستراتيجية خاصة لاستعمالها في حلّ المسائل الأكثر تعقيداً .

سنقوم بتوضيح هذا المسار أو هذا الخط بواسطة مثل بسيط يُمثّل طريقـة لضرب عناصر مصفوفتين (matrix) .

نص المسألة : لنفترض المصفوفة A (matrix) تتألف من عدد n من الأسطر وعدد n من الأعمدة ، والمصفوفة B وتتألف من عدد n من الأسطر وعدد P من الأعمدة . المطلوب أن نحسب نتيجة ضرب المصفوفتين :

$$C = A \times B$$

التحليل التصاعدي :

المخطط التالي يدل على طريقة تشكيل الضرب المصفوفي لكل i وإلى كل j ، هنا :

$$C^i_j = \sum_{K=1}^{N} A^i_k \times B^k_j$$

حسابة العنصر C^i_j تتم على الشكل التالي :

$C(I, J) \leftarrow 0$

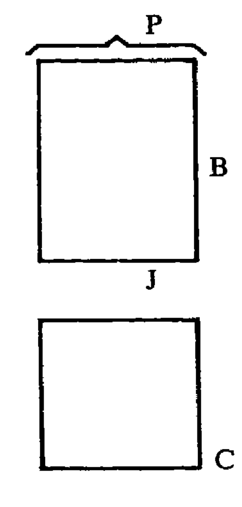
لكل متغيرة I من 1 إلى N كرِّر

$$C(I, J) \leftarrow C(I, J) + A(I, K) * B(K, J)$$

أو :

. FOR I = 1 TON Repeat

$$C(I, J) \leftarrow C(I, J) + A(I, K) * B(K, J)$$



n

A

لحسابة السطر I ، سنقوم بتغيير مؤشر الأعمدة J .

لكل J متغيرة من 1 إلى P

$C(I, J) \leftarrow 0$

كرِّر   لكل K متغيرة من 1 إلى N

كرِّر

$C(I, J) \leftarrow C(I, J) + A(I, K) * B(K, J)$

```
FOR J = 1      TO P    Repeat
C (I, J) ← 0
For K = 1 TO N  repeat

        C (I, J) ← C (I, J) + A (I, K) * B (K, J)
```

هكذا ، لحسابة المصفوفة C كاملة ، سنستعمل الخوارزم التالي :

```
Var A, B, C Matrix
Begin
C (I, J) = 0
FOR I = 1 TON Repeat
FOR J = 1 TON Repeat
FOR K = 1 TON Repeat

C (I, J) ← C (I, J) + A (I, K) + B (K, J)
      end
    end
  end
```



الفصل الثالث

المواضيع البسيطة ، الأنواع ، التعابير ، كتابة النتائج

يعالج الخوارزم مواضيع وأشياء ، يقوم بإجراء الأفعال عليها . تختلف المواضيع باختلاف نطاق إستعمالها ، وهي عادة :

- المواضيع البسيطة :
- الأعداد . مثلاً : 3.145, 1978
- السمات . مثلاً : 'A', '9', 'C', ...
- القيم الجبرية أو المنطقية . مثلاً F, T (false, true)

ولكي نستطيع معالجة هذه الأعداد أو السمات أو المواضيع بشكل عام ، يجب التصريح عنها في بداية البرنامج ، يتضمن التصريح عن المواضيع معلومات عن نوعيتها ، وحجمها .

مثلاً : « صباح الخير سيد أحمد ، إني بحاجة إلى كلغ سكر وكلغ خبز » .

نلاحظ في هذا المثل :

- تحديد طبيعة المواضيع أو الأشياء المطلوب معالجتها (شراؤها) . مثلاً : المطلوب : رز، خبز ، سكر . . .
- عدم إستعمال هذه المواضيع أو هذه الأشياء بشكل عشوائي . مثلاً : يجب أن نزن الرز ، وشرب الكولا ، الخ . . .

أي إن لكل نوع من هذه المواضيع ، عمليات خاصة بها .

هكذا ، يُدعى نوع (type) الإلتقاء بين :

- طبيعة المواضيع (رز ، سكر ، أعداد صحيحة ، حقيقية . . الخ) .
- العمليات المرتبطة بها (وزن ، غسيل ، جمع ، ضرب الأعداد . . . الخ) .

3.1 ـ الأنواع الأساسية والعمليات المرتبطة بها.

نستطيع تمييز أربعة أنواع أساسية للأعداد ، هي :

- ـ الصحيحة integer .
- ـ الحقيقية ، real .
- ـ المنطقية أو البولية ، boolean .
- ـ السمات character .

أ ـ أنواع الصحيح (entier, integer)

قيمة هذه الأنواع تُمَثِّـل الأعداد الصحيحة .

التعبير عن هذه الأنواع :

نستعمل عادة الترميز العشري لتمثيل هذه الأعداد .

الرموز المستعملة هي : 9, 8, 7, 6 , 5, 4, 3, 2, 1, 0

وبإمكاننا إستعمال أي من التعابير المستعملة لتمثيل المعلومات العددية أو الرقمية ، أي الأعداد ، كالترميز الثنائي في النظام الثنائي بواسطة الأرقام 1, 0 .

العمليات الجارية على هذه الأنواع هي العمليـات العاديـة (الجمع ، الـطرح ، الضرب ، القسمة ، و mod) .

ب ـ النوع الحقيقي real type

قيمة هذا النوع عبارة عن أعداد حقيقية ، وتُمَثِّـل هذه الأنواع (الأعداد) بواسطة الترميز العشري غالباً ، أو في الترميز الثنائي ، الثماني ، أو السادس عشـري في بعض الأحيان مثلاً :

1987, 153.50, 0.0012

النقطة تُمَثِّـل الفاصلة العشرية في هذه الأعداد .

العمليات الجارية على هذه الأنواع هي نفسها تلك التي تجري على الأعداد الحقيقية في الجبر . مثلاً : الجمع ، الطرح ، الضرب ، القسمة . .

تمثيل تكويد هذه الأعداد بلغة باسكال .

const ; e = 1.71828 ; blanc = '' ; N = 3

وعلى العكس فإن العدد

N1 = N + 1 غير مسموح .

3.2 ـ التعريف عن المتحولات والأنواع

3.2.1 ـ التعريف عن المتحولات والأنواع الحقيقية والصحيحة بلغة باسكال

```
Var i : integer ;
s, y, z : real;
```

ـ i : عبارة عن معرِّف للمتحولة المستعملة (identificator) . وكلمة integer تدلّ على إن المتحولة i هي من نوع صحيح .

ـ x, y, z : عبارة عن أسماء متحولات أو معرِّفات مصرَّح عنها على إنها من النوع الحقيقي .

3.2.2 ـ التعريف عن المتحولات والأنواع المنطقية أو البولية

تأخذ المتحولات المنطقية قيمتان فقط : صح ، خطأ (false, true) . ترمز إلى هاتان القيمتان بواسطة الأعداد المنطقية : T أو .TRUE. ,F أو .FALSE. .

العمليات : تُستعمل العمليات المنطقية العادية (و ، أو ، لا) NOT ، OR ، AND .

يجري التصريح عن هذه المتحولات بلغة باسكال كما يلي :

```
VAR X, Y : Boolean;
```

3.2.3 ـ النوع سمات : (CAARACTER OR STRING)

يتألف هذا النوع من جميع السمات العادية المستعملة في اللغة : الأحرف ، الأرقام ، السمات التي يُقال عنها بأنها خاصة (نقطة (.)، فاصلة (،) ، الفراغ (   ) ، الدولار (S) ، الخ . .) .

التكويد : تُحاط السمة بواسطة أبوستروف (' ') . مثلاً :

'A', '9', '#'

في الخوارزميات ، يُرمز إلى السمة « فراغ » (blanc,space) على الشكل التالي : ''

العمليات : نستعمل فقط عمليات المقارنة على السمات ، كالعمليات التالية : = (يعادل) ، ≠ (يختلف) .

3.2.3 ـ الأنواع المرقمة (enumurated type)

نجد هذه الأنواع في لغة باسكال (أو في لغة آدا) .

مثلاً : VAR JOUR : (Lundi, mardi, mercredi , jeudi) ;

المتحولة JOUR (يوم) ، يمكن أن تأخذ فقط واحدة من القيم الموجودة ضمن الهلالين : أي من Lundi إلى Jeudi .

تعريف هذا النوع :

التعريف في النوع المرقم يسمح بربط إسمٍ معين بالنوع ، وذلك في كل مرّة نحتاج فيها إلى هذا النوع . مثلاً : لبناء نوع إنشائي مركّب عند التصريح عن المتحولات .

مثلاً :

```
type response = (yes, No, may be);
chifre = '0' .. '9';
```

التصريح التالي :

```
Var dig : chiffre;
```

سيستبدل التصريح :

```
var dig: '0' .. '9' ;
```

في كل مرّة نرغب فيها ببلوغ النوع chiffre المصرَّح عنه كنوع في بداية البرنامج .

3.3 ـ التعابير expression

3.3.1 ـ حسابة وتقييم التعابير

الطريقة الأكثر شيوعاً لمعالجة المواضيع من إعداد وثوابت ومتحولات ، تقوم على إدخالها في عملية معينة حسابية أو منطقية . تُستعمل لذلك المؤثرات البسيطة (المؤثرات المنطقية NOT ، AND ، OR) على متأثر واحد (مؤثرات آحادية (مؤثرات بمتأثر واحد (single operand operator)) ، أو على متأثرين (مؤثرات ثنائية (two operands operator)) . هذه المتأثرات (operand) تنتمي إلى أحد الأنواع المعرَّفة أعلاه .

لنفترض a وb هي مواضيع (object) بمؤثر (operator) آحادي ، $\bar{\wedge}$ هو مؤثر ثنائي ، وexp1 ، exp2 عبارة عن تعابير . فإذاً :

$$a,\ \underline{\vee}a,\ a\ \bar{\wedge}\ b,\ exp1\ \bar{\wedge}\ exp2$$

عبارة عن تعابير مرتبطة بعمليات تدل عليها المؤثرات $\underline{\vee}$ و$\bar{\wedge}$.

لرفع الإبهام عن التعابير وتوضيح العمليات ، نقوم بوضع أولوية (priority) لكل مؤثر . هذه الأولوية تسمح بترتيب المؤثرات فيما بينها . القواعد التالية تُحدِّد نظام الحساب .

يجب البدء أولاً بتقييم وحسابة أو تنفيذ تلك العملية التي تتمتع بأولويـة أكبر من العمليات التالية المتقاربة .

مثلاً : لحسابة 2 / 5 + 3 نبدأ بمؤثر القسمة
وحسابة 5/2 وبعد ذلك نقوم بجمع نتيجة القسمة هذه مع 3 .

ـ في حالٍ تساوي الأولوية ، نبدأ بالحساب من اليسار إلى اليمين .
مثلاً : لحسابة : 5 − 2 + 3
نبدأ بواسطة 2 + 3 وبعد ذلك نطرح من النتيجة 5 .

ـ من الممكن دائماً حصر العمليات الجزئية بداخل أهلّـة كي نستطيع إحترام نظام أولوية المؤثرات عند إجراء الحساب .

3.3.2 ـ المؤثرات بلغة باسكال

أ ـ المؤثرات الجبرية .

− الطرح

+ الجمع

* الضرب

/ القسمة

div القسمة

mod modulo

ب ـ المؤثرات المنطقية :

and, Or, NOT

ج ـ المؤثرات العلائقية .

= يعادل

<> يختلف عن

< أقل من

> أكبر من

<= أقل أو يساوي

>= أكبر أو يساوي .

د ـ الدالات الجبرية

abs (x) ، القيمة المطلقة من x

sqr (x)	مربع x (x^2)
sin (x)	جيب x
cos (x)	جيب تمام x
arctg (x)	
exp (x)	الدالة الأسية
ln (x)	لوغاريتم x
sqrt (x)	الجذر التربيعي من x

فلنشير هنا إلى إن حسابة دالة الرفع إلى أس معين تتم على الشكل التالي :

$$x^y = ex\,p(y) * \ln(x)$$

هـ ـ الدالات الجذرية

succ (x)

pred (x)

ard (x)

chr (x)

و ـ دالات التبادل

trunc (x) = القيمة الصحيحة من x إذا كانت $x \geqslant 0$ ، أو القيمة الصحيحة من abs (x) إذا كانت $x < 0$.

مثلاً :

trunc (17.986) = 17

trunc (– 4.3) = – 4

round (x) = العدد الصحيح الأقرب إلى x .

round (17.61) = 18

round (17.42) = 17

ز ـ الدالات المنطقية .

odd (x) تعادل true إذا كان التعبير من النوع صحيح x هو مفرد، وfalse إذا كان التعبير الصحيح نفسه مزدوج ..

من الممكن أن نعتبر إن :

odd (x) = abs (x) mod 2 = 1

eof تدل على نهاية السجل

eoln(f) تدل على نهاية السطر من النص f .

إذا جرى إهمال f ، فذلك يعني أن النص هو سجل الإدخال أي ما يدخل بواسطة الأداة الطرفية (input file) .

3.4 ـ كتابة النتائج (Writing)

تقوم الآلة الحاسبة الجزئية بعرض متواصل ومنتظم لجميع النتائج الجزئية الوسيطية . ولكن باستعمال حاسب كبير تكون النتائج مختلفة ومتعددة ، وعملية عرضها عند الإخراج (على طابعة أو على شاشة) يجب أن يتم بشكل واضح مُفصَّل وجلي . وفي حدود الخوارزم ، نعتبر إن عملية الكتابة تتم بشكل متواصل على أسطر دون الاهتمام بحجم هذه الأسطر وبعدد السمات المطلوب عرضها أو طباعتها . هكذا ، يجب إستغلال سهولة الطباعة كي نستطيع تحضير النتائج بشكلٍ جيد .

أمر الكتابة هو :

WRITE (a1, a2, ... an);

أو :

إكتب (a1, a2, ... an) ;

حيث :

ـ ai يمكن أن تُمثّل :

.. تعبير جبري أو منطقي .

ـ سلسلة من السمات محصورة بداخل أبوستروف (' ') .

ـ الأمر « على السطر » (on line) يؤدي إلى الطباعة على سطر جديد تالي .

ـ عدد

مثلا :

WRITE (' the square of 4 = ' , 4 × 4);

يؤدي إلى طبعة النتيجة التالية :

the square of 4 = 16

ترجمة هذا الأمر بلغة باسكال هو :

WRITE (e1, e2, ..., en);

يعادل :

```
begin
    write (e1);
    write (e2);


    write (en);
end.
```

هذا الأمر يؤدي إلى كتابة e1, e2, ... en على نفس السطر .

الإجراء writeln يؤدي إلى إنهاء السطر ، أي كتابة قيمة المتحولات على نفس السطر والعودة إلى بداية السطر الجديد . هكذا فالتعليمة :

```
writeln (e1, .. en);
    begin
        write (e1);
        .
        .
        write (en);
        writeln
        end
```

تعادل الأوامر التالية :

مثل 1 :

إكتب عدد يساوي n من النجوم (*) على نفس السطر :

```
for x : = 1 to n do
    write (' * ');
    writeln
```

مثل 2 :

إكتب p * n مرّة الرمز '—' على نفس السطر ، وذلك على مجموعات ، تحتوي كل واحدة على عدد P من الرمز ـ ، وتنفصل عن الأخرى بواسطة الرمز + . أي على الشكل التالي :

$$\underbrace{___}_{P} + ___ + ___ +$$

```
for i : = 1 to n do begin
    for j : = 1 to p do
        write ('—');.
        write (' + ')
        end;
    writeln
```

هناك ثلاثة حالات قد تظهر خلال كتابة النتائج .

ـ التعليمة write (e) ، تكتب قيمة e على شكل سمة ، إذا كانت e عبارة عن سمة معينة .

ـ ;write (e:n) ، تكتب n − 1 فراغ ، وبعد ذلك قيمة المتحولة على شكل سمة (في الحالة التي تُمثَّل فيها e سمة معينة) .

مثلاً :

write ('a', 'a': 2) → a a

ـ e عبارة عن عدد صحيح .

ـ ; write (e) : تكتب التمثيل العشري للمتحولة e .

ـ ;write (e:n) ، تكتب التمثيل العشري للمتحولة e ، مسبوقة بعدد من الفراغات الضرورية للحصول على عدد n من الرموز في المجموع .

مثلاً :

write (12: 2, − 12 :0 4 , 127: 1) ;

هذه التعليمة تؤدي إلى كتابة :

12   − 12   27

ـ e عبارة عن عدد حقيقي .

ـ write (e) ، تكتب قيمة e بشكل عدد حقيقي بفاصلة متحركة (أي هو write (e:n) حيث n تتعلّق بالحاسب المستعمل) .

ـ write (e: n) : تسمح بكتابة قيمة e على شكل عدد بفاصلة متحركة مسبوق بعدد من الفراغات يسمح بتغطية عدد يساوي n من السمات في المجمع . وإذا كان الحقل من n سمة هو غير كافٍ ، فسيتم توسيعه .

ـ write (e: n: d) ، يكتب قيمة e على شكل عدد حقيقي بفاصلة ثابتة وبعدد مساوٍ له من

الأرقام للقسم الكسري (العشري) من الجزء العشري (mantisse) ، وهناك فراغات تسبق العدد لتغطية مكان n سمة . وإذا كان الحقل المؤلف من n سمة هو غير كافٍ فسيتم توسيعه .

- e ـ عبارة عن متحولة من السمات .
  - ;write (e) ، تُكتب السمات x من السلسلة حسب ورودها في المتحولة .
  - ;write (e:n) ، يُطبع العدد n من السمات الأولى التي تؤلف السلسلة e ، على أن تُسبق بالعدد اللازم من الفراغات لتغطية الحقل المؤلف من n موقع (أو n سمة) إذا كان n > x .
- e ـ عبارة عن متحولة منطقية
  - ;write (e) ، تكتب السلسلة 'true' أو السلسلة 'false' .
  - ;write (e:n) ، تقوم بتكبير الحقل من n من السمات .

3.5 ـ إدخال المعلومات أو قراءة المعطيات Reading

الأمر المستعمل لقراءة المعلومات يبدو على الشكل التالي :

Read (a1, ... an);

أو إقرأ (an .. a1) ؛

وتترجم بلغة باسكال بواسطة التعليمة التالية :

```
Read (a1, .. an);
read ln (a1 .. an);
```

في الأمر الأول يقرأ الحاسب المتحولات an .. a1 الواحدة بعد الأخرى ولا ينتقل إلى السطر التالي إلا إذا تجاوز طول جميع المتحولات an..a1 طول سطر واحد . بينما في التعليمة Readln فإن الحاسب يقرأ سلسلة المتحولات الموجودة بداخل الهلالين وينتقل بعدها تلقائياً إلى بداية السطر التالي .

3.5 ـ التسمية ، التتالي ، القراءة ، الإلحاق

قد نقوم خلال تحليل المسألة بتقسيمها إلى مسائل ـ ثانوية . يجب أن نعمل على حلّ هذه المسائل الثانوية حسب نظام ترتيبها وورودها وتواليها . النتائج الحاصلة من حلّ كل مسألة ثانوية يمكن أن تُستعمل كمعطيات لحلّ المسائل الثانوية اللاحقة . من المناسب إذاً ، ولكي نتمكن من معالجة هذه النتائج ، أن نقوم بتسميتها وهذا يتم بواسطة المعرِّف (identificator) .

من الممكن إذاً تسمية هذه الثوابت ، مثلاً ربط المتحولة Pi بالثابتة 3.14 (Pi = const 3.14) ، أو أيضاً تسمية معطيات الخوارزم التي نحصل عليها بواسطة أحد أوامر القراءة .

3.5.1 ـ التسمية Nommation

يعني المعرِّف (identificator) إحدى القِيم الناتجة عن الحساب . يُضاف إلى كل معرِّف مجموعة من المميزات التالية :

- إسم المعرِّف يجب أن يكون وحيداً ، يجب تسمية نتيجتين مختلفتين بواسطة مُعرِّفين مختلفين .
- نوع المعرِّف (type) : ويدل على المجموعة التي تنتمي إليها القيمة المعنية بواسطة هذا المعرِّف .

تُحدَّد هذه المميزات عند التصريح عن المعرِّف .

النحو الخاص بالتصريح :

(Syntax declaration)

t identificator is v

حيث t : تدل على نوع المعرِّف .
v : إسم المعرِّف .

أمثلة :

```
real Pi is 3.1415926535
real twpi is 2 * pPi
bool test is true
int nb mois is 12
```

الدلالة :

- يجب أن تكون قيمة v متوافقة مع النوع المعلن t .
- بعد تحديد المعرِّف، لا يمكننا تغيير قيمته .
- يجب التصريح عن كل معرِّف مستعمل في الخوارزم في بداية البرنامج .

أمثلة :

```
type chiffre = '0' .. 'q';
var digit : chiffre ;
```

أي : تأخذ المتحولة digit إحدى قيم chiffre

const pi = 3.14

أي إن :

ـ Const ـ تدل على إن المتحولة هي ثابتة .
pi ـ إسم المعرَّف أو إسم الثابتة .
3.14 ـ قيمة المعرِّف أو الثابتة .

3.5.2 ـ التوالي (sequentiality) :

لنفترض إن المسألة P تنقسم إلى المسائل الثانوية S1، S2، S3 ، وإن هذه الأخيرة يجب أن تُنفَّـذ حسب هذا الترتيب ، نستعمل الرمز « ; » كفاصـل بين هـذه الاجراءات أو المعالجات . هكذا ، ستُحدَّد P بواسطة الكتابة التالية :

Start S1; S2, S3 END

مثلاً :

إكتب الخوارزم الذي يحسب مساحة وحجم كرة بشعاع يعادل 1.24 سم .

مساحة الكرة تحسب بواسطة المعادلة :

$$S = 4\pi . R^2$$

حجم الكرة يعادل :

$$V = 4\pi R^3/3$$

نلاحظ أن حسابة قيمة V نحصل عليها من S بواسطة الصيغة V =S * R/3 مما يسمح لنا بتقليص عدد العمليات .

هكذا ، فبالإمكان كتابة الخوارزم على الشكل التالي :

```
Real Pi is 3.1459;
Real radius is 1.24;
real surface is 4 * Pi * radius * radius;
real volume is surface * radius/3;
write (' the surface of sphère = ' surface,
                    'volume = ' , volume)
END.
```

3.5.3 - الإلحاق COLLATERALITY

لا تعالج جميع عمليات الحساب التابعة للخوارزم بالضرورة بشكل متسلسل ، لأنها قد تكون مستقلة . الحالة الأكثر كلاسيكية هي تلك الخاصة بتقييم التعابير . فلنفترض التعبير الجبري a * b + c/d ، تفرض قواعد الأولوية أن نحسب a * b وc/d قبل إجراء عملية الجمع . وعلى العكس ، لا يوجد أي شيء يدل على ضرورة تنفيذ a * b قبل c/d .

هكذا ، لنفترض المسألة P التي من الممكن أن تُقسَّم إلى ثلاثة مسائل مُستقلّة S1 S2, S3 . هذه المسألة يمكن أن تُمثَّل بواسطة S1, S2, S3 حيث الفاصلة (،) تدل على الالحاق ، على عكس النقطة والفاصلة (؛) التي تعني التوالي .

مثلاً :

لنفترض عددين حقيقيين a وb للقراءة حسب ورودها ، نرغب بحسابة نتيجة الضرب p = a × b ، والقسمة a/b ، والمجموع p + q . بالإمكان أن نكتب الخوارزم على الشكل التالي :

```
start
     real a is Read real;
     read b is Read real;
     real b is a * b, real q is a/b;
     write (' the product of' , a, 'and', b,
           'Equal': , p, ' and the quotient' , q,
           'the SUM of p and q = : ', p + q)
end
```

الفصل الرابع

التكرار ، المتحولات ، التخصيص
(Iteration, Variables, Assignements)

4.1 ـ الهدف :

الأفعال : إحسب المبلغ المدفوع لكل عامل من شركة معيّـنة .

إحسب الأعداد أل n الأولى من سلسلة من الأعداد . إجراء معالجـة على جميـع عناصر المتجة (vector) الخ . .

إذا كان عدد عمليات التكرار كبيراً ، فمن الصعب إعادة كتابة نفس الخوارزم لعدد n من المرّات . سيكون من غير الممكن كتابته إذا كانت n غير معروفة مسبقاً وفي البداية . مثلاً : إذا كان ذلك يتعلّـق بمعالجة مجموعة من المعطيات « حتى نصل إلى آخرها ولا يبقى منها شيئاً » . يجب إذاً أن نتمكّـن من تحديد مدى عملية تكرار الفعل ، أي التكرار ، الذي وبعد التصفير والإعداد ، يتتابع حتى تحدث حادثة معينة : هذه هي حادثـة إنهاء التكرار ، الذي نحدِّده في الخوارزم بواسطة أحد الشروط .

في عملية التكرار ، نلاحظ إنه من الجاري والشائع أن نحسب العناصر المتتالية في إحدى السلاسل المحدَّدة على أنها تكرارية ، أي حيث العنصر u_n يتعلّـق بالعنصر u_{n-1} :

$$u_n = f(U_{n-1})$$

وإذا كانت القيم الوسيطة ... u_1, u_2 ليست محفوظة ، نقوم بتمثيلها وتحديدها بشكل متتالٍ بواسطة موضوع مُوحَّـد يُدعى متحولة (Variable) . هذه القيم ... $u_1, u_2,$ أو هذا الموضوع الجديد أي المتحولة يجب أن يكون موضوع تصريح ، أي يجب أن يُصرَّح عنه وعن نوعيته وحجمه (مثلاً : التصريح عن عدد العناصر في أحد الجداول وعن نوعيتها) .

وبشكل عام ، نستطيع إعطاء المتحولة قيمة أولية .

هناك عملية تدعى التخصيص (assignement) ، وهذه العملية تسمح بتغيير قيمة

المتحولة حسب رغبة المبرمج .

في أغلب الأحيان نستعمل إحدى المتحولات (كمؤثر) لتحديد شرط إنهاء عملية تكرار الفعل .

4.2.1 ـ عملية التكرار « حتى حادثة أعمل »

تكتب هذه العملية بواسطة الاجراء التالي :

```
To évenement DO
    A1;
while condition sort by evenement
    A2
Continue
```

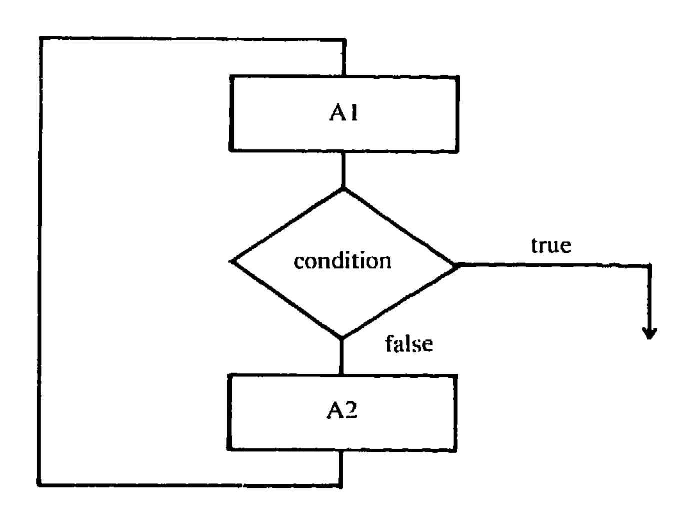
ـ evenement : عبارة عن معرّف (identificator) معيّن أو إسم متحولة .

ـ condition : عبارة عن تعبير منطقي يُمثّل شرط معين للتنفيذ ، نتيجة هذا الشرط هي صحّ (true) أو غلط (false) .

ـ A2، A1 : عبارة عن أفعال (عمليات ، تعليمات ، أوامر . .) . واحدة من هـذه الأفعال قد تكون فارغة بدون أثر (أي بدون نتيجة) .

ـ while ... to : (حتى . . . عندما) هما عبارة عن مفاتيح أوامر التكرار .

دلالة التكرار يمكن أن تتمثّل بواسطة المخطط التالي :



هذه التعابير (حتى . . . إفعل ، عندما شرط إفعل) يمكن أن تترجم بلغة باسكال بواسطة الأوامر التالية :

```
1) while C do
     begin
     statements
     end.
2) repeat
     statement
     .
     .
statement
UNTIL C
```

C ـ عبارة عن تعبير منطقي أو شرط أو حادثة معينة .

ملاحظة : من الممكن في لغة باسكال أن نستعمل عكس الشرط لمتابعة التكرار while NOT C ، أي إذا كان الشرط غير صحيح نتابع التكرار .

مثلاً : الجذر التربيعي لعدد إيجابي a يعادل حدود السلسلة التكرارية $u_i = (u_{i-1} + a/u_{i-1})/2$ هكذا ، فإذا أسمينا حدود السلسلة l فعند ذلك سيكون معنا :

$l = (l + a/l)/2$ ، أي إن $l^2 = a$

لذلك ولنحصل على l بدقة معينة تعادل epsilon يجب وقف عملية التكرار لقيمة المؤشر x عندما تصبح قيمة التعبير $\frac{|a - Ux^2|}{a}$ أصغر من epsilon .

هكذا ، فالبرنامج الذي يحسب الجذر التربيعي باستعمال طريقة نيوتن هو :

```
program Newton (input , output);
{ حساب الجذر التربيعي باستعمال طريقة نيوتن }

Var u: real;            { سلسلة التقاربات. }
    a: real;            { العدد الذي نبحث عن جذره }
    epsilon : real;     { الدقة }
begin
```

```
      epsilon : = 0.0001;
read (a) ; u: = 1;
repeat
      u: = (u + a / u)/2
until (abs (a − u * u) / a) < epsilon;
writelin (n)
end.
```

4.2.2 ـ التصريح عن المتحولات

التصريح عن المتحولات يتم على الشكل التالي :

Var t identificator

identificator ـ وتعني اسم المتحولة .

t ـ نوع المتحولة

Var كلمة مفتاح (Key word) ، وتُمثِّل الايعاز أو الأمر بالتصريح عن المتحولة .

معنى هذا التصريح ، أن المتحولة identificator تعني أو تدل على قِيم من نوع t .

أو بكلمة أخرى ، إن المتحولة 0 المُسماة identificator هي من نوع t .

لاعطاء قيمة أولية للمتحولة ، نكتب ما يلي :

Var t identificator unit expression

التعبير expression هو من نوع t ويُمثِّل القيمة الأولية للمتحولة المسماة identificator .

أمثلة .

Var INT total;

ـ النوع INT : مختصر كلمة INTEGER (صحيح) ويُمثِّل نوع المتحولة الذي رمزنا إليه بالحرف t (type) ، total ويُمثِّل المعرِّف identificator .

معنى هذا التصريح إن المتحولة total هي من النوع INTEGER أو الصحيح .

Var real moyenne unit 0;

المتحولة moyenne ذات القيمة الأولية 0 هي من النوع real أو الحقيقي .

من الممكن جمع التصريحات عن عدة متحولات (بدون إعداد أو إعطاء قيمة أولية لها) ، وذلك إذا كانت هذه المتحولات من نفس النوع ، وعلى الشكل التالي :

Var INT nb1 , nb2, nb3;

4.2.3 ـ التخصيص (assignement , Affectation)

تتم عملية التخصيص على الشكل التالي :

identificator : = expression

: : ـ الرمز = : يدل على عملية التخصيص .

ـ التعبير expression ، ويمكن أن يكون عبارة عن ثابتة أو معادلة جبرية أو منطقية ، ويجب أن يكون نوعه متطابقاً مع نوع القيم المعنية بواسطة المتحولة identificator . بعد عملية التخصيص يصبح التعبير expression هو القيمة الجديدة للمتحولة identificator .

مسألة : معنا سلسلة من الأعداد الحقيقية المعرَّفة بواسطة المعادلة التالية :

$n_0 = a$

$n_i = f(u_{i-1})\ i > 0$

المطلوب حسابة العنصر الأول u الذي يجاوب على الشرط $C(u_i)$. الخوارزم المناسب لذلك يمكن أن يكتب على الشكل التالي :

```
begin
    Var real u init 9;
    for u DO
        while C(u) Sort by u
            u: = f (u)
    continue;
    write ('the result is ' , u)
```

مثلاً : لحسابة $\sqrt{}2$ ، يمكن أن نستعمل السلسلة التالية :

$u_0 = 1$

$-2/u_{i-1})/2\ (i > 0) u_i = (u_{i-1}\ - 2/u_{i-1})/2 \qquad (i > 0)$

القيمة التي نبحث عنها تعادل حدود u عندما تتجه i نحو اللانهاية ، وبإمكاننا إعتبار شرط الكفاية من وجهة نظر الدقة المطلوبة الشرط التالي :

$$epsilon = | u^2_i - 2 | < 0.001$$

والخوارزم المناسب سيدو على الشكل التالي :

```
begin
                                        {حسبان الجذر التربيعي للعدد 2 }
real epsilon is 0.001;
Var real u init 1;
    for square root DO
      while abs (u ↑ 2 – 2) < epsilon
        sort by square root;
      u: (u + 2/u)/2
      continue;
    write ('the square root of 2 = ', u)
END
```

4.3 ـ التوسيع Extension

في بعض الحالات ، يمكن أن تكون حوادث التوقف نفسها مختلفة . مثلاً ، البحث عن عنصر في مجموعة من العناصر يمكن أن ينتهي بنجاح ـ إذا وجدنا العنصر المطلوب ـ أو بخسارة ـ إذا لم نجد هذا العنصر بعد البحث عنه في كامل المجموعة ودون نتيجة .

وفي جميع الأحوال ، فالأفعال الواجب إتخاذها في هذه الحالات أو ردّات الفعل على النجاح أو الخسارة يمكن أن تكون مختلفة .

الشكل العام للتكرار يسمح بتحديد هذه الحالة :

بداية

حتى الحادثة 1 . . . أو الحادثة 2 . . . أو الحادثة n أفعل

....

AK;

عندما الشرط cond k أخرج بواسطة الحادثة i ؛

....

Al;

عندما الشرط cond l أخرج بواسطة evj ؛
إفعل
مخرج ev1 : فعل s1
. . .
مخرج evj : فعل sj
مخرج evn : فعل sn
نهاية التكرار
صياغة هذا الخوارزم باللغة الانكليزية تبدو على الشكل التالي :

```
FOR ev1 or ... or evj ... or evn DO
....
Ak;
while cond k sort by evi;
....
Al;
while cond l sort by evj;
....
DO

output ev1 : action s1
....
output evj : action sj
....
output evn : action sn

end of iteration
```

وبالإمكان ترجمة هذا الخوارزم باللغة الفرنسية على الشكل التالي :

```
Jusqu'à ev1 ou ... ou evj .... ou evn faire
....
Ak;
```

```
quand cond k sortir par evi;
....
Al;
quand cond l sortir par evj;
....
DO

sortie ev1: action s1
....
sortie evj : action sj
....
sortie evn : action sn
fin iteration
```

سيتم تنفيذ فعل واحد هو Sj : أي الفعل الذي يناسب الحادثة evj التي أثارت المخرج Sj .

مسألة :

البحث عن أحد العناصر بداخل لائحة من الأعداد (Liste) : لنفترض لائحة تحتوي على أعداد صحيحة وإيجابية ، وتنتهي بالعدد صفر . سنبحث فيها إذا كان أحد الأعداد المعروفة مسبقاً موجوداً ضمن اللائحة .

الخوارزم المذكور يقوم بالبحث بداخل اللائحة حتى يجد العدد المطلوب .

لنفترض أن العدد الذي نبحث عنه هو NB .

المتحولة التي تمثل العدد الموجود في اللائحة هي : value .

المتحولة التي تمثل العدد الغائب هي : value abscent

الأعداد الموجودة في اللائحة هي nb list .

```
Begin
INT NB is readint;
Var INT nb list init readint;
```

```
for value OR value abscent DO
while nb list = 0 sort by value abscent;
while nb list = NB sort by value;
        nb list = readint
    Continue
output value : write ('the number is present')
output value abscent : write ('the number is abscent);
end.
```

المتحولات المستعملة في هذا الخوارزم هي :

- readint : عدد صحيح يدخل إلى المكنة بواسطة القراءة (مثلاً : إدخل عدد بواسطة لوحة ملامس الشاشة) . وهو يمثل الأعداد المقروءة من اللائحة .

- NB : هو العدد الذي نبحث عما إذا كان موجوداً في اللائحة . هذا العدد تقرأه المكنة من الخارج ، ويتم إدخاله أو قراءته بواسطة لوحة الملامس .

- value abscent : إذا لم نجد العدد ، فستكون القيمة غائبة ، وبالتالي العدد NB الذي نبحث عنه في اللائحة غير موجود بداخلها .

ملاحظة : الخروج من التكرار المتداخل .

عندما تكون عمليات التكرار متداخلة ، فليس من المسموح الخروج من عملية التكرار إلا بواسطة إحدى الحوادث المتصلة بعملية التكرار هذه .

هكذا فالصياغة التالية هي غير صحيحة .

```
FOR e1 DO
  ....
    FOR e2 on e3 DO
    ....
      while C sort by e1;
    DO
output e2;
    outpute3;
end of interation
```



الفصل الخامس

الاختيار (CHOIX)

5.1 ـ الهدف :

في أكثر الحالات ، لا يتم تنفيذ الأفعال إلا إذا تأمنت بعض الشروط :

ـ لنفترض المعادلة التالية :

$$ax^2 + bx + c = 0$$

فإذا كانت القيمة $\sqrt{b^2 - 4ac}$ سلبية ، تكون المعادلة بدون جذور . أما إذا كانت $\sqrt{b^2 - 4ac}$ تساوي صفر ، تكون المعادلة بجذر واحد . وفي الحالة الأخيرة ، إذا كانت $\sqrt{b^2 - 4ac}$ إيجابية ، تكون المعادلة بجذرين مختلفين .

ـ إذا أصبح مخزون سلعة معينة معادلة لقيمة دنيا ، يجب وضع طلبية جديدة .

عمليات الاختيار هذه مرتبطة بشروط معينة بواسطة تعبير بولي أو منطقي . فإذا كانت قيمة هذا التعبير (صح) true ، فهذا سيؤدي إلى تنفيذ معالجة معينة أو أفعال مرتبطة بهذا التعبير . لنفترض إن Ci عبارة عن عدد i من الشروط المختلفة ، وإن Ti عبارة عن عدد من المعالجات المرتبطة بـ الشروط Ci . فعملية الاختيار تكتب كما يلي :

```
Case
C1 → T1,
C2 → T2
......
Cn → Tn
end of case.
```

الشروط Ci هي عبارة عن تعابير منطقية ، والمعالجات Ti عبارة عن أفعال يتم تنفيذها

حسب قيمة الشروط Ci .

- يتم تقدير وحسابة التعابير الشرطية المنطقية Ci بشكل متكامل ومتحد . هذه الشروط يجب أن تكون تبادلية ومقتصرة ، أي هناك شرط واحد على الأكثر بقيمة true (حقيقة) .

$$\forall\, i, j, i \neq j, Ci \text{ AND } Cj = \text{false}$$

عندما يكون الشرط Ci « حقيقة » ، سيتم تنفيذ المعالجة أو الاجراء المناسب المرتبط به .

ملاحظة : إذا رغبنا بتنفيذ الفعل Tn + 1 عندما تكون الشروط C1, ... Cn غلط (false) . نكتب ما يلي : end of case التي تعني نهاية الحالات .

```
Case
C1 → T1;
C2 → T2;
....
Cn → Tn;
else → Tn + 1
end of case
```

لنفترض إن جميع الشروط Ci غلط :

- فإذا جرى توقع معالجة خاصة بعد else ، فسيجري تنفيذ هذه المعالجة Tn + 1 قبل الانتهاء من التعليمة case ، وإلا تنتهي case بدون تنفيذ أي فعل أو إجراء .

تسمح لغة باسكال بالتعبير عن هذه الحالة بواسطة التعليمة التالية :

```
CASE expression - test OF
     valeur 1 : instruction 1;
     valeur 2 : instruction 2;
     .
     .
     .
valeur n: instruction n;

END;
```

عندما تكون قيمة التعبير expression-test تعادل القيمة valeur i ، سيتم تنفيذ التعليمة i (والتي من المحتمل أن تكون فراغاً) . يجب أن تكون القيم i عبارة عن ثوابت ، كما يمكن أن تجمّع عدة قيم إذا كان الاجراء أو المعالجة المناسبة هي نفسها .

إذا كانت قيمة التعبير expression-test لا تعادل أية قيمة حاضرة من valeur i ، فسينتهي البرنامج إلى خطأ .

وتعتبر هذه التعليمة أشد فعالية من نظيرتها في لغة فورتران (GOTO) وبازيك (ON) ، لأن المنتقي ليس بالضرورة هو صحيح والقيم ليست بالضرورة متتالية .

مثلاً : إحسب جذور المعادلة $ax^2 + bx + c$ ($a = 0$) .

```
Begin
    real a is read real;
    real b is read real;
    real c is read real;
    real delta is b ↑ 2 − 4 * a * c;
case
    delta < 0 → write ('without + real solution'),
    delta = 0 → write (' one solution, x = ', − b/2 * a)
    delta > 0 → write (' two solution: ' , on line,
                                    ' x1 = ', ( − b + √ delta) = (2 * a))
' x2 = ', ( − b − √ delta)/(2 * a))
end of case
END;
```

مسألة

اكتب بلغة باسكال البرنامج الذي يقوم بحلّ المسألة التالية :

التسجيلة المُميزة للزبون تحتوي بالتحديد على حرف أبجدي يُحدِّد الحسم الموافق للزبون حسب القاعدة التالية :

'A' حسم 10%	'C' حسم 10%
'B' حسم 5%	'D' حسم 5%

وذلك إذا كان المبلغ يزيد على 1000 ليرة ، وإلا دون حسم .

```
Var LC : CHAR;
    MONTANT, APAYER : REAL;
....
CASE LC OF
    'A', 'C' : APAYER : = 0.9 * MONTANT;
    'B'      : BEGIN
    IF MONTANT > 1000.0
    THEN APAYER : = 0.95 * MONTANT
    ELSE APAYER : = MONTANT
    END;
'D' : APAYER : = 0.95
END;
```

ـ المتحولة MONTANT : تعني المبلغ الاجمالي .

ـ المتحولة APAYER : تعني المبلغ بعد الحسم أو المبلغ الواجب دفعه .

5.2 ـ التعليمات الانشائية IF .... THEN ... ELSE

```
IF C THEN T1
    ELSE T2
END IF
```

إذا شرط C إذن T1

وإلا T2

أو بالفرنسية نكتب ما يلي :

```
Si C alors T1
    Sinon T2
```

ـ T2، T1 : عبارة عن تعليمات أو إجراءات أو أفعال .

ـ C : عبارة عن شرط معين على أساسه يتم تنفيذ التعليمات T1 أو T2 . هذا الشكل يناسب التعليمة التالية في لغة باسكال .

```
IF C THEN
        instruction 1
        ELSE
          instruction 2;
     instruction 3;
```

أي إذا كـانت قيمة الشـرط المنطقي C هي (صحيحـة) «true» فسيتم تنفيـذ التعليمة instruction 1 وبعد ذلك التعليمة instruction 3 . أما إذا كانت قيمة الشرط C هي false فسيتم تنفيذ التعليم instruction 2 وبعد ذلك instruction 3 .

مثلاً :

```
IF A > B THEN
        WRITELN ('A is Great than B')
        ELSE
        WRITELN ('A is less than B');
WRITELN ('END OF SEARCH');
```

مسألة :

حلّ معادلة الدرجة الأولى $Ax + b = 0$ ،

إذا كانت قيمة A مختلفة عن صفر $A = 0$ ، فالجذر سيعادل $X = -A$

أما إذا كانت قيمة A تعادل 0 $(A = 0)$ ، فسيتم طباعة «impossible» أو غير محدَّد «UNDETERMINATED» حسبما إذا كانت قيمة B تعادل صفر أو مختلفة عن صفر .

```
IF A = 0 THEN
     IF B = 0 THEN WRITELN ('UNDETERMINATED')
          ELSE WRITELN ('IMPOSSIBLE')
     ELSE
          BEGIN
          x : = - B / A:
          WRITELN ('The root is = ', x)
          END;
```

الفصل السادس

الجداول ARRAY

يجمع الجدول مجموعة معينة من القيم التي تمتاز بنفس الخصائص وتنتمي إلى نوعٍ معين : وبالإمكان تسمية جميع القيم ، أو إستعمال تعبير أو تحديد دقيق لواحدة من القيم المخزنة في الجدول أو أحد عناصره بواسطة عملية تأشير معينة تدل على العنصر أو القيمة .

نجد مثلاً : هذا النوع من المسائل في ترقيم الأحرف الأبجدية . فإذا أطلقنا الاسم Alphabet على مجموعة الأحرف ، فمعنى ذلك ، إن :

ـ Alphabet ستدل أو تعني مجموعة الأحرف الأبجدية .

ـ العنصر [14] Alphabet يدل على العنصر رقم 14 وهو الحرف 'N' .

ـ العدد 14 يُدعى مؤشر .

6.1 ـ التصريح عن الجداول

للتصريح عن الجداول نستعمل بشكل عام التعابير الخوارزمية التالية :

أ ـ التصريح عن جدول من الثوابت :

[inf : sup] t identificator is v

ب ـ التصريح عن جدول من المتحولات :

[inf : sup] var t identificator { init v }

وهذا يعني :

ـ identificator : عبارة عن إسم الجدول بكامله .

ـ t : وتدل على نوعية عناصر الجدول (type)

ـ V : هو ترميز لعدد n من القيم في الجدول على الشكل التالي :

(V1, V2, ... Vn) ، حيث Vi عبارة عن تعبير من النوع t

ـ وإذا أدخلنا init إلى التصريح ، فهذا يسمح لنا بإعداد وتهيئة متحولات الجدول identificator إلى القيمة V ، أي إن عناصر الجدول ستأخذ كقيمة أولية القيم V .

ـ inf : sup : تمثّل حدود الجدول الدنيا والعليا أو أبعاد الجدول .

مثلاً :

```
[ 1: 26 ] CHAR alphabet is ('a', 'b', 'c', ... 'z');
[ 1: n ]  var real v
```

المتجه (Vector) V يحتوي على n عدد حقيقي .

بينما الجدول alphabet ، يتألف من سطر واحد و26 عامود (يُمثّل متجه (Vector)) ، ويحتوي على رموز أبجعددية ، عبارة عن الأحرف الأبجدية z... b, a .

6.1.2 ـ التصريح عن الجداول بلغة باسكال

يبدو التصريح عن الجداول بلغة باسكال على الشكل التالي :

```
VAR V: ARRAY [ 1...3 ] OF REAL;
    W : ARRAY [ 1..3 ] OF REAL;
```

يحتوي الجدول ذو الاسم W وV على ثلاثة عناصر حقيقية من النوع real .

ـ real هي نوع عناصر الجدول (t) .
ـ V ـ identificator وتُمثّل معرِّف الجدول أو إسمه .

ولكن إذا كان هناك عدة جداول (متجهات) من نفس النوع ، ومطلوب معالجتها ، فمن الممكن إنشاء نوع متجه بثلاثة أبعاد (three dimensional vector) .

وذلك على الشكل التالي :

```
TYPE VECT 3 = ARRAY [ 1...3 ] OF REAL;
      VAR V : VECT 3;
          W : VECT 3;
```

6.2 ـ معالجة عناصر الجدول بالتتالي

النحو المستعمل لصياغة تعليمات المعالجة يبدو كما يلي :

```
identificator [ index ]
```

ـ المعرف identificator يُمثِّل إسم الجدول .

ـ index عبارة عن تعبير صحيح حيث القيمة يجب أن تنتمي إلى الفسحة [inf, sup] .

ـ التعبير[index]identificator يعني عنصراً من الجدول يُشار إليه بواسطة المؤشر الخاص به (index) في الجدول .

مثل 2 :

مداولة المتجه (Vector manipulation) .

لنفترض المتجه V الذي يحتوي على الأعداد الحقيقية (V = (a1, a2, ... an . إذا كان هو معرِّف الجدول الذي يُمثِّل المتجه هو V ، فالتعبير[i] V يعني العنصر رقم i من الجدول ، أي العنصر ai .

الخوارزم التالي يحسب نتيجة ضرب المتجهين :

```
Begin
      INT n is 5;
        [ 1:0 n ] real V1 is (2.0, 3.0, 8.0, 0.1, 1.0);
        [ 1: n ] real V2 is (−5.0, 0.8, 3.0, 2.5, − 1.0);
Var real scolar product INIT 0.0;
Var int index init 1;
TO all elements DO
while index Sn sort by all elements;
scolar product + : = V1 index × V2 index ;
index: = index + 1
continue

write (scolar product)
end.
```

6.3 ـ الحلقة « إلى ـ حتى » (FOR)

عندما نرغب بتكرار المعالجة لعدد محدَّد ومعروف مسبقاً من المرّات ، فمن الأفضل استعمال الحلقة FOR من الحلقة TO. فلنأخذالمثل الذي يعالج الضرب الساكن (Scalar product) بين متجهين .

باستعمال الحلقة « حتى » (TO) يبدو الخوارزم على الشكل التالي :

```
Var real PS init 0.0;
Var int ind init 1;
TO vector DO
PS + : = [ ind ] × V2 [ ind ] ;
ind : = ind + 1
while ind > n sort by vector
DO;
```

أما بواسطة الحلقة FOR ، فسيبدو الخوارزم على الشكل التالي :

```
Var real PS init 0.0;
FOR ind from 1 by step 1 TO n DO
PS + : = V1 [ ind ] V2 [ ind ]
DO;
```

هكذا ، فالنحو المستعمل لتمثيل الحلقة « إلى ـ حتى » يبدو على الشكل التالي .

إلى i من e1 بخطوة e2 حتى e3
إفعل S
إفعل .

أو بالفرنسية :

```
pour i depuis e1 pas e2 jusqu'à e3
     faire S
     faire S
     fait
```

أو بالإنكليزية :

```
FOR i from e1 step e2 TO e3
     DO  S
     DO
```

حيث :

ـ i : تدعى متحولة التحكُّم بالحلقة ؛ وهي عبارة عن متحولة داخلية في الحلقة . لذلك لا

نقوم بالتصريح عنها . إضافة لذلك لا يجب أن يتم تغيير قيمة هذه المتحـولة داخـل الحلقة ، كما ويجب أن تكون متحولة من النوع « صحيح » (integer) .

ـ e1، e2، e3 : عبارة عن تعابير جبرية تحسب مرةٍ واحدة قبل البدء بعملية التكرار ، أو هي ثوابت .

ـ e1 : هي القيمة الأولية التي تأخذها متحولة التحكُّـم بالحلقة i ، وعندما تكون e1 = 1 فبإمكاننا إلغاء الجملة from e1 .

ـ e2 : تدعى خطوة الحلقة ، وإذا كانت e2 = 1 ، فبإمكاننا إهمال الجملة by step e2 .

ـ e3 : عبارة عن القيمة النهائية التي تبلغها المتحولة i .

ـ 5 : عبارة عن سلسلة من التعليمات التي تُشكِّل جسم الحلقة . بلغة باسكال ، تترجم الحلقة « إلى ـ حتى » بواسطة التعليمة التالية :

```
FOR i: = exp1 TO exp2 DO
    BEGIN
      S
    END;
```

مسألة :

إحسب الضرب الساكن بين مُتجهين .

الصيغة الرياضية لضرب المتجهات U وV هي :

$V.V = A\ U_i V_i$

البرنامج :

```
Const N = 20;
Type DIMENSION = 1... N;
    VECT = ARRAY [ DIMENSION ] OF REAL;
VAR U, V : VECT;
        UV : REAL;
      I   : DIMENSION;
```

```
    UV : 0;
FOR I : = 1 TO N DO
BEGIN
UV : = UV + V [ I ] * V [ I ]
END;
```

6.4 ـ الجداول بعدة أبعاد (Multidimensioned Array)

عندما تكون عناصر الجدول هي نفسها عبارة عن جداول ، عند ذلك نتكلم عن جدول بعدة أبعاد . مثلاً ، الصورة المرسلة بواسطة القمر الاصطناعي هي شبيهة بجدول (جدول ببعدين) حيث كل عنصر يمثّل القيمة الضوئية (مثلاً 0 = أبيض ، 100 = أسود) لنقطة من الصورة . وإذا جرى تحديد الصورة بواسطة 128 × 128 نقطة ، فالتصريح عنها يتم على الشكل التالي :

[1: 128 , 1: 128] var int image

integer - int O

أي أن الجدول image عبارة عن جدول ببعدين من الأعداد الصحيحة ويحتوي على 128 × 128 عنصراً أو عدداً .

هكذا فالتصريح عن الجداول بعدة أبعاد يتم على الشكل التالي :

أي التصريح عن جدول من الثوابت بعدد n من الأبعاد يتم كما يلي :

[inf1: sup1, inf2: sup2, ... inf n: supn] t identificator is v

ب ـ التصريح عن متحولة من نوع جدول بعدد n من الأبعاد :

[inf1: sup1, inf2: sup2, ..., inf n: sup n] vart identificator init { init v }

ج ـ البلوغ إلى عنصر من جدول بعدة أبعاد يتم حسب الطريقة التالية :

identificator [index1, index2, ..., index n]

مثلاً : لنفترض جدول الثوابت التالي :

	1	2	3
1	1	20	10
2	3	4	5

فالتصريح عنه يتم على الشكل التالي :

```
[ 1: 2, 1:3 ] table is ((1, 20, 10), (3, 4, 5))
```

إذن العنصر [2,1] Table يعادل القيمة 3 .

ملاحظة :

infl: supl يدل على عدد الأسطر ، وبالتالي فعدد الأسطر يعادل supl – infl .

sup2 – inf2 يعادل عدد الأعمدة في الجدول .

مثلاً :

إحسب مجموع مصفوفتين من الثوابت A وB بأبعاد 4 × 3 .

```
Begin
[ 1:3, 1:4 ] int A is ((25, −1, 0,3), (12, 3, −8,0), (1, −8, 2, −1));
[ 1:3و 1:4 ] int B is ((3, −2, −4, 8) , (6, −8, 0, 0), (1, 1, 1, 1));
FOR i TO 3 DO
    For j TO 4 DO
    write (A [ i, j ] + B [ i, j ] )
    DO ;
  DO;
DO
```

بلغة باسكال ، يتم التصريح عن المصفوفة كما يلي :

```
TYPE name = ARRAY [ 1..N, 1..N ] OF TYPE;
    TYPE = integer, real, char
```

مسألة :

نتيجة ضرب المصفوفتين B, A ، حيث :

```
A = [ 1..N, 1..N ]
B = [ 1..N, 1..N ]
```

تعادل المصفوفة C بحيث يعادل العنصر C_{ij} :

$$C_{ij} = \sum_{k=1} A_{ik} B_{kj}$$

لحل هذه المسألة سنستعمل حلقتين مختلفتين ، لكل منهما مؤشر خاص i وj .

لاجتياز جميع العناصر C_{ij} ، سنُثَبِّت i وj داخل الحلقات ونحسب C_{ij} ، هذا العنصر الأخير هو عبارة عن مجموع ، من هنا سينتج حلقة ثالثة بمؤشر جاري هو k .

البرنامج :

```
PROGRAM PRODUCT
TYPE MATRICE = ARRAY [ 1..N, 1..N ] OF REAL:
     VAR A, B, C : MATRICE
       I, J, K : 1.. N;
     BEGIN
       FOR I : = 1 TON DO
         FOR J : = 1 TON DO
           BEGIN
C          [ I, J ] : = 0.0;
         FOR K : = 1 TO N DO
C          [ I, J ] : = C[ I, J ] + A [ I, K ] × B [ K, J ]
         END
END
```

6.5 ـ الجداول المتراصة بلغة باسكال (PACKEDARRAY)

تستعمل وتعالج الجداول المتراصة بنفس الطريقة التي تعالج بها الجداول العادية مع فارق يكمن في طريقة ترتيب خزن عناصر الجدول .

كما نعرف ، فإن السمة (Character) تشغل بايتة واحدة (8bits = 1byte) لخزنها في الذاكرة .

لنفترض بأننا نستعمل مكنة بكلمة طولها 32 بتة (4 bytes) .

فإذا صرَّحنا عن الجدول بالطريقة العادية ARRAY [. . .] OF CHAR ، سيتم تخزين كل سمة في كلمة مختلفة ، ولكن إذا صرَّحنا عن نفس الجدول كمتراص PACKED ، فسيتم تجميع السمات بشكل يتم فيه تخزين كل أربعة بايتات في كلمة واحدة .

التصريح عن الجدول المتراص يتم كما يلي :

```
Type ALFA = PACKED ARRAY [ 1.. 10 ] OF CHAR;
```

إذاً ، تؤمن الجداول المتراصة توفير في الذاكرة ، ولكن ذلك يتم على حساب سرعة الحساب : هكذا ، فلبلوغ عنصر معين من الجدول المُتراص ، يجب أن نجد الكلمة التي يوجد بداخلها هذا العنصر ، وبعد ذلك يجب أن نبحث عن العنصر بداخل الكلمة . بالإمكان رصّ وبعثرة عناصر الجداول ، وهذه هي الاجراءات المسماة PACK وUNPACK التي تسمح بذلك .

مثلاً :

```
VAR A: ARRAY [ 1.. N ] OF TYPE DEA;
    PA : PACKED ARRAY [ 1..N ] OF TYPE DEA;
```

الاجراء UNPACK (PA, A, I) يعادل الحلقة التالية :

```
OR K = 1 TO N DO
    A [ K – 1 + I ] : = PA [ K ] ;
```

أما الإجراء PACK (A, I, PA) فيعادل الحلقة :

```
    for k = 1 TO N DO
PA [ K ] : = A [ K – 1 + I ] ;
```

6.6 ـ النوع تسجيلة record type

6.6.1 ـ النوع record

الجدول هو الوسيلة الوحيدة لتجميع عدد n من العناصر من نفس النوع . مثلاً : أرقام عمل الزبون في العشر سنوات الأخيرة ، أو أسماء جميع الزبائن . . . ولكن من الضروري تجميع العناصر التي تختلف بأنواعها ، وهذه هي تحديداً ، الحالة بالنسبة للتسجيلات التي تؤلف سجلاً (فايل file) معيناً ، مثلاً ، يجب أن نحفظ لكل زبون ملفاً يحتوي على المعلومات التالية :

● رقم الزبون	Numero	Numero
● إسم الزبون	Nom	Nom
● عنوان الزبون	adresse	ADRESSE
● الكود البريدي	Post code	cpville
● التمثيل الذي يشغله في العالم	representant	NUMEREP

● إذا كان له حسم	Remise	REMISE
● أرقام عمله في السنوات السابقة	Chiffre d'affaire	CAPREC
● رقم عمله في السنة الحالية	chiffre d'affaire	CA COURS

هكذا ، فلغة باسكال تحتوي على النوع reord الذي يؤمن التصريح عن هذه المعلومات المختلفة ، والتصريح عن ذلك ، هو كما يلي :

```
TYPE CLIENT = RECORD
NUMERO        : INTEGER;
NOM           : PACKED ARRAY [ 1..10 ] OF CHAR;
ADRESS        : PACKED ARRAY [ 1..30 ] OF CHAR;
CPVILLE       : PACKED ARRAY [ 1.. 20 ] OF CHAR;
NUMEREP       INTEGER;
REMISE        : BOOLEAN;
CAPREC        : REAL;
CACOURS       : REAL;
```

هكذا فمن الممكن التصريح عن سجل الزبائن (client) CLI ، كما يلي :

```
VAR CLI : CLIENT;
```

وبإمكاننا أن نبلغ إلى كل عنصر من CLI بواسطة الطريقة التالية :

```
CLI . NOM : = '   DUPONT
IF CLI. REMISE THEN ...
```

وبالإمكان خلط ذلك مع التأشير :

```
TYPE ENTREPRISE = ARRAY [ 1..50 ] OF CLIENT;
    VAR E1 : ENTREPRISE;
```

في هذه اللحظة ، يكون رقم العمل في السنة الحالية للزبون الخامس من الشركة E1 (entreprise) ، هو :

```
E1 [ I ] .CACOURS
```

والسمة الأولى من اسم هذا الزبون ، هي :

```
E1 [ 5 ] . NOM [ 1 ]
```

مسألة :

يحتوي كل سطر من الطلبية على ما يلي :

- نص (إسم السلعة) (Libellé) ، بطول 20 سمة .
- سعر الوحدة أو السلعة (Unit price) ، عدد حقيقي .
- عدد السلع (nombre) ، صحيح (integer)
- المبلغ (montant) ، عدد حقيقي (real) .

6.6.2 ـ التعليمة with

من الملاحظ إن إعداد وتهيئة التسجيلة record بالطريقة الكلاسيكية المذكورة أعلاه هي عملية مضجرة ، هكذا فتخصيص قيم معينة لعناصر التسجيلة يجب أن نكتب ما يلي :

```
CLI . NUMERO : = 1000;
CLI.NOM : = '    DUPONT         »;
CLI. ADRESSE : = 'BEYROUTH'
CLI. CPVILLE : = '          75010 TYPE
CLI. NUMREP : = 10;
CLI. REMISE : = TRUE;
```

ولكن باستعمال التعليمة with تصبح عملية الإعداد (initialisation) وإعطاء قِيم لعناصر التسجيلة أمراً سهلاً ، وهي تسمح بتفادي عملية تكرار CLI في كل مرَّة .

هكذا نكتب :

```
With CLI DO
BEGIN
    NUMREP : = 1000;
    NOM : = 'AHMED'
    .
    .
    .
CACOURS : = ...
END;
```

هذه الطريقة هي مفيدة للغاية عندما يكـون عندنـا عدة مستـويات في التسجيلة RECORD ، أي عدة مصرِّفات للعنصر الواحد كما نعرف في المثل التالي للحصول على

إسم الزبون من الشركة ENTREPRISE الخاصة بالمصنع USINE في القسم SERVICE من السجل CLI :

```
ENTREPRISE . VISINE . SERVICE. CLI. NOM : = AHMED
```

يجب إذاً كتابة التعليمة التالية باستعمال with .

```
with ENTREPRISE, USINE, SERVICE, CLI DO
    BEGIN
      NOM : = ...
END;
```

6.6.3 ـ التسجيلات المتحولة

من الممكن أن لا تكون جميع التسجيلات بنفس التركيبة . لنفترض مثلاً ، لائحة عمال للمعالجة . هذه اللائحة ستحتوي مؤكداً على إسم وتاريخ ولادة العامل ، ومن ثم على إشارة تدل على الوضع العائلي : «C» أعزب ، 'M' ، متزوج ، 'D' مطلق ، 'V' أرمل . في الحالة التي يكون فيها العامل أعزباً ، أي 'C' ، ستكون المعلومات كاملة ، ولكن في الحالة التي يكون فيها متزوجاً ، أي إن التسجيلة تحتوي على الحرف 'M' ، فالتسجيلة ستحتوي على إسم الزوجة إضافة إلى تاريخ الزواج . أما في الحالة التي يكون فيها العامل مطلقاً أو عندما يكون أرملاً ، فستحتوي التسجيلة على تاريخ الطلاق أو تاريخ الترمل .

التصريح عن هكذا تسجيلة يتم على الشكل التالي :

```
Type NNN = PACKED ARRAY [ 1..10 ] OF CHAR;
   DATE = RECORD
       JR : 1... 31;
       MS : 1... 12;
       AN : 0.. 995
   END;
EMPL : RECORD
       NOM : NNN;
       PRENOM : NNN;
```

```
        DTNAIS : DATE;
        CASE SITFAM : CHAR OF
'C': (   );
'M': (NOM CONJOINT: NNN;
     PREN CONJOINT: NNN;
     DATE MAR        : DATE);
'D', 'V' : (DTVD: DATE)
END;

VAR EMPLOYE : EMPL;
```

المتحولات المستعملة في البرنامج تعني ما يلي :

:JR يوم (JOUR)

:MS شهر (MOIS)

:AN سنة (Année)

:EMPL عامل (EMPLOYE)

:nom إسم العائلة

: PREN الإسم PRENOM

: DTNAIS تاريخ الولادة (DATE NAISSANCE)

: 'C' أعزب

: 'M' متزوج

: 'D' مطلق

: 'V' أرمل

: NOM. CONJOINT إسم عائلة الزوجة

: PREN. CONJOINT إسم الزوجة

: DATEMAR تاريخ الزواج

: DTVD تاريخ الطلاق أو تاريخ الترمل

التصريح الأخير

VAR EMPLOYE : EMPL;

يعني إن السجل EMPLOYE يتألف من تسجيلات من نوع EMPL . فلنشر إلى

إن CASE تحدِّد الحقل SITFAM (الحالة العائلية) ونوعه . تعليمة الاستعمال يمكن أن تكون من نوع :

```
READ (EMPLOYE);
  IF EMPLOYE . SITFAM = 'M' THEN
    WRITELN ('DATE MARIAGE' , EMPLOYE . DATEMAR)
      ELSE
    WRITELN (' NOM MARIE');
```

فلنشر إلى إنه إذا كان العامل غير متزوج ، فلا يوجد أي بلوغ إلى DATE MAR . يجب دائماً إختبار الحقل SITFAM لنعرف الحقل الذي نستطيع بلوغه .

6.7 ـ النوع مجموعة TYPE SET

تفهم المجموعة بشكل حدسي في الرياضيات . فهي عبارة عن تجميع للعناصر . مثلاً : لنفترض العناصر A, B, C ، فبإمكاننا أن نُشكِّل مجموعات مختلفة من هذه العناصر . مثلاً :

[ ] المجموعة فراغ

[A,B] ، [A, C] ، [B, C] ، [A, B, C]

هكذا ، فإذا كانت العناصر A, B, C تُشكِّل نوع TYPE بلغة باسكال:

```
Type ELEMENT = (A, B, C);
```

والمجموعات المذكورة أعلاه هي عبارة عن مواضيع من نوع :

```
Type ENS = SET OF ELEMENTS
```

من هنا نرى ، إنه من الممكن إنشاء مجموعات بلغة باسكال ، عندما يكون عندنا مجموعة من المواضيع أو الأشياء تنتمي إلى نفس النوع .

هذه المجموعات تُشكِّل النوع SET OF الذي يجمع جميع المجموعات التي من الممكن تشكيلها مع العناصر من النوع الأساسي .

6.7.1 ـ العمليات على المجموعات

جميع العمليات العادية من علم المجموعات هي معرَّفة بلغة باسكال . ولكن هذه

العمليات لا تُطبَّـق إلا على مجموعات من نفس النوع ، أي SET OF من نفس النـوع الأساسي .

لقد رأينا حتى الآن عملية التخصيص (assignment) .

```
Type BASE = (B1, B2, ...., BN);
     ENS = SET OF BASE
VAR E1, E2, E3 : ENS;
     A1, A2, A3 : BASE;
```

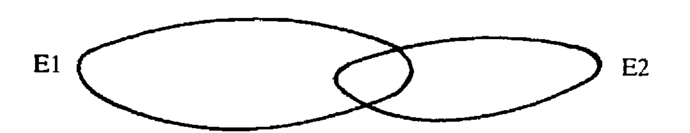
عمليات التخصيص التالية :

```
E1: = [ B1, B2 ] ;
E2: = [ A1 ] ;
```

هي عمليات صحيحة .

أ ـ عملية الاتحاد (Union)

تكتب مثلاً كما يلي : E3: E1 + E2 ، وبلغة باسكال ، هذه العملية تُنْشيء مجموعة جديدة مؤلفة من جميع العناصر التي تنتمي إلى E1 أو إلى E2 .



مثلاً :

```
E1: = [ B1, B2 ] ;
E2: = [ B3, B4 ] ;
E3: = E1 + E2
```

ب ـ التقاطع (intersection)

وتُكتب بلغة باسكال كما يلي :

```
E3 = E1 × E2
```

وتؤدي إلى إنشاء مجموعة من العناصر المشتركة فيما بين المجموعتين E2, E1 . هكذا مثلاً :

$$E1 \times E2 = [\ B3\ ]$$



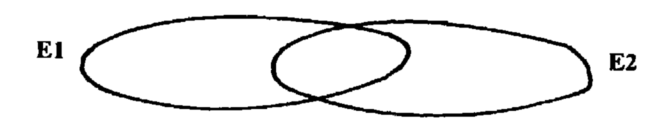
ج ـ التتام (Complement)

وتُكتب كما يلي :

E3 : E1 − E2

والنتيجة تعادل مجموعة العناصر من E1 الغير موجودة في E2 . هكذا مثلاً ، حسب المعطيات المصرَّح عنها سابقاً : نرى إن :

$$E3 = E1 - E2 = [\ B1\ ]$$



6.7.2 ـ العمليات العلائقية relational operations

نتيجة هذه العمليات هي بولية أو منطقية ، وهنا أيضاً لا تنطبق هذه العمليات إلا على مجموعات من نفس النوع .

أ ـ التعادل : E1 = E2 هذه العملية هي صح TRUE ، إذا كانت المجموعتين متشابهتين أي كلٍ منهما تحتوي بالضبط على نفس العناصر .

ب ـ عدم التعادل (NOT EQUAL) : E1 < > E2 هي TRUE ، إذا كانت إحدى المجموعتين E1 وE2 تحتوي على الأقل على عنصر واحد غير مشترك .

● التداخل (Inclusion) : E1 < = E2 أو E2 > = E1 هي TRUE ، إذا كانت E2 تحتوي على الأقل على جميع عناصر E1 .

أ ـ الانتماء : هذه العلاقة هي محدَّدة بين عنصر من نوع أساسي ومجموعة .

النتيجة هي منطقية ، مؤشر علاقة الانتماء IN هو كلمة محجوزة . هكذا فالعلاقة التالية :

A1 IN E2 هي TRUE

إذا كان العنصر A1 ينتمي إلى المجموعة E2 .

لهذه العلاقة تطبيقات مريحة . مثلاً ، إذا صرَّحنا كما يلي :

```
VAR X : CHAR;
```

فالتعليمة :

```
IF X IN [ 'A', 'E', 'I', 'O', 'V', 'Y' ]
```

تفحص فيما إذا كان العنصر X هو حرف ساكن داخل ضمن مجموعة الأحرف المذكورة بعد IN .

مسألة :

ترجم بلغة باسكال علاقة التداخل E1 < E2 . قراءة المجموعة تتم بقراءة متتالية لجميع عناصرها (إذا كانوا من نوع نموذجي) ، واتحادها المتتالي من خلال المجموعة فراغ هو ممكناً .

```
TYPE CARAC = SET OF CHAR;
    VAR CAR : CAR A C;
        C: CHAR;
BEGIN
    CAR: = [    ];                    { إعداد المجموعة فراغ }
    REPEAT
        READ (C);
        CAR : = CAR + [ C ]           { نضيف السمة المقروءة }
    UNTIL EOF
END;
```

طباعة المجموعة CAR ستتم على الشكل التالي ، إذا إفترضنا إنها محصورة بالأحرف :

```
typc:CARAC = SET OF 'A' ... 'Z';
     VAR CAR : CARAC;
          C : 'A' .. 'Z';
     BEGIN
          FOR C : = 'A' TO »Z' DO
            BEGIN
            IF C IN CAR THEN
              WRITE (C)
            END;
END;
```

الفصل السابع

الاجراءات (PROCEDURE)

7.1 ـ مدخل

أهمية هذا الموضوع هي كبيرة ، فالإجراء يسمح للمبرمج بمعالجة المسألة دون الاهتمام ، وفي الوهلة الأولى ، لتفاصيل الاجراء .

الاجراء هو شبيه « بالدالات » (function) التي نراها في الرياضيات ، فاستعمال الدالة max في تعبير رياضي معين ، يسمح بإمكانية قراءة جيدة لهذا التعبير ، ولكن يجب قبل ذلك أن نقوم بالتصريح عن هذه الدالة max . وفي البرمجة نستطيع القول إن الاجراء يُمثِّل خوارزم معين، وجسمه يعالج متغيرات شكلية (argument, formal parameter) . تنفيذ هذا الخوارزم يتم بعد نداء لهذا الإجراء من داخل برنامج رئيسي : فمعطيات هذا الخوارزم هي عبارة عن متغيرات حقلية متناسبة مع المتغيرات الشكلية . يقوم الاجراء عادة ، بإعادة نتيجة معينة إلى البرنامج الرئيسي ، ولكنه قد يُستعمل فقط بمساعدة البرنامج الأخير في حلّ المسألة المطروحة دون الحاجة إلى كتابة جسمه لعدة مرّات ، وبالتالي فقد لا يُعيد أية قيمة . في الحالة الأولى ، النداء هو عبارة عن متأثر (operand) من تعبير جبري (مثلاً max (b, c) في التعبير (a + max (b, c) .

الفائدة المنهجية لهذه المواضيع تقوم بشكل أساسي ، وفي المرحلة الأولى ، على تحويل المسائل ـ الثانوية إلى إجراءات ، وعلى معالجة المسألة المطروحة وكأنها عبارة عن إجراءات ثانوية . الفائدة الأخرى ، الأكثر عملانية والتي تتصل بالفائدة الأولى ، تكمن فيما إذا كان من الواجب حلّ المسألة ـ الثانوية لعدة مرّات ، وفي كل مرّة باستعمال متغيرات وسيطية فعلية مختلفة (effectiv parameter) ، فعند ذلك يمكننا إستعمال نفس التعليمات من جسم الاجراء بدلاً من كتابتها في كل مرّة في جسم البرنامج الرئيسي ، وبشكل عام يؤدي إستعمال الاجراء إلى تحسين إمكانية قراءة الخوارزم وتعديله وتطوره .

7.2 ـ النحو المستعمل في التصريح عن الاجراء ونداءه

7.2.1 ـ التصريح عن الاجراء

يأخذ التصريح عن الاجراء أحد الأشكال التالية :

```
1) PROC identificator is
     { fixe (a1, a2, ..an } result type
     Begin
     Body of procedure
     end

2) PROC identificator is
     { fixe (a1, a2, ... an } { mod (a1, a2, ... an) }
     Begin
          Body of procedure
          END
```

ـ في الحالة الأولى ، يؤدي نداء الاجراء إلى إنتاج قيمة من النوع المحدَّد type . هذا النوع قد يكون Integer ، real ، boolean أو character .

ـ identificator : هو إسم يعني ويدل على الاجراء .

ـ المتحولات الشكلية $a_1, a_2, ... a_n$ عبارة عن متحولات شكلية (formal parameter) وتكون على الشكل التالي :

$t_1, x_1, t_2\ x_2 ...$

حسب الحالة 2 :

```
PRO C change i j is
    fixe (INT i, int j) mod (Var INT x)
    Begin
        IF X = i THEN x : = j end of il
END.
```

7.2.2 ـ صياغة تعليمات النداء

لنداء الاجراء نكتب بداخل البرنامج الأساسي التعليمة التالية :

identificator { $(a_1 ... a_n)$ } { $(b_1 ... b_n)$ }

حيث :

ـ identificator : عبارة عن إسم الاجراء المستعمل في النداء عند التصريح عنه .

ـ $a_1 \ldots a_n$, $b_1 \ldots b_n$ تُمثّل المتحولات الفعلية (effectiv parameters) .

ـ المتحولات الفعلية ($a_1 .. a_n$) تناسب المتحولات الشكلية المحدَّدة بعد الكلمة fix عند التصريح عن الاجراء ، أما المتحولات ($b_1 \ldots b_n$) فتناسب المتحولات الشكلية المحدَّدة بعد mod .

مثلاً : (A1)

إستعمال الاجراء max من المثل السابق :

```
proc max is
    fixe (INT x, y) result INT
      Begin
        IF x > y THEN result x
                        ELSE result y fund of if
```

حيث x_1 ، x_2 . . . تُمثّل المعرَّفات الشكلية للمتحولات الشكلية . وt_1 ، t_2 . . . تُمثّل أنواع هذه المعرَّفات المطلوبة من المتحولات الفعلية . هذه الأنواع تشترط إستعمال المتحولات الشكلية في جسم الاجراء .

● جسم الاجراء (body of procedure) : عبارة عن الأفعال أو التعليمات والأوامر المطلوبة لحلّ المسألة الثانوية .

في الحالة (1) ، يؤدي تنفيذ الاجراء إلى إنتاج قيمة معينة من النوع المحدَّد بعد الكلمة result عند التصريح عن الاجراء . هذه القيمة هي عبارة عن النتيجة النهائية لتنفيذ الاجراء . يدعى هذا النوع من الاجراءات بالاسم دالة (function) .

جرى إدخال fixe وmod لتجميع المتغيرات الشكلية التي تتمتع بنفس الخصائص .

مثلاً :

حسب الحالة الأولى :

```
PROC max is
    fixe (INT x, INT y) result INT
    BEGIN
```

```
        IF x > y THEN result x
                        ELSE result y end of if
END.

INT a is read int
INT b is read int
    write (' the max of', a, 'and', b, 'is:', max (a, b))
END
```

مسألة :

لنفترض سلسلة من الأعداد مؤلفة من اعداد صحيحة إيجابية . المطلوب إيجاد سلسلة جديدة من الأعداد ناتجة عن الأولى بحيث يتم إستبدال جميع الأعداد التي تعادل 10 في السلسلة الأولى بالعدد 11 .

مثل A2 : سنقوم باستعمال الاجراء change من المثل السابق في الفقرة 7.2.1 .

```
Begin
{ use the procedure change i j from paragraph 7.2.1 }
proc change i j is
    fixe (INT i, INT j) mod (Var INT x)
    Begin If x = i then x : = j end of if end;
    INT a is 10, INT b is 11,
    INT flag is - 1,
    Var INT NB init read int
        TO research DO
        while Nb = flag sort by research;
          change (a, b) (Nb);
          write (Nb);
          Nb: = read Int
continue;
    write (Nb)
END;
```

المتحولات المستعملة في هذا الخوارزم هي :

ـ إسم الاجراء هو change ، هذا الاجراء يستبدل i بالعدد j . المتحولة a تعادل 10 ، والمتحولة b تعادل 11 .

ـ j, i : عبارة عن متحولات شكلية للإجراء .

ـ b, a : عبارة عن متحولات فعلية في الإجراء .

ـ Nb : هو العدد المقروء والذي عليه تقسم عملية المقارنة . هذا العدد يعادل دائماً العدد الداخل read int ، أو العدد الصحيح من اللائحة الذي تقرأه المكنة . هذا العدد (read int) تستلمه المكنة وتخصصه للمتحولة Nb .

ـ التعليمة to end of list DO

وتعني حتى نهاية اللائحة ، أي حتى يتم إدخال جميع الأعداد .

ـ read int

ـ المتحولة flag : وتعادل 1− ، وهي إشارة جرى إدخالها لتدل على نهاية اللائحة .

ـ التعليمة : while Nb = flay sort by research

وتعني ، عند بلوغ الاشارة 1 − التي تدل على نهاية سلسلة الأعداد ، يجب الخروج من السلسلة وإنهاء عملية البحث (research) .

ـ عند قراءة كل عدد ، يجري نداء الاجراء chang ، لاستبدال العدد a بالعدد b (أي إستبدال 10 بـ 11) .

ـ يجري كتابة العدد Nb = 11 في موقع العدد 10 .

ـ بعد ذلك تجري قراءة العدد التالي من السلسلة .

Nb : = read int

ـ يُتابع العمل حتى بلوغ نهاية السلسلة : Continue —

7.2.3 ـ الدلالة :

المسألة تقوم على تعريف الخوارزم المعادل لنداء الإجراء . دون الدخول في التفاصيل ، باستطاعتنا القول إن هذا الخوارزم هو مُركّب من جسم الإجراء المسبوق بتصريحات تؤدي إلى التناسب بين معرّفات المتحولات الشكلية والقيم المقدمة من المتحولات الفعلية : هذه التصريحات تحتوي على المتحولات الشكلية على يسار الكلمة IS وعلى يمينها تحتوي على المتحولات الفعلية المناسبة لها .

مثلاً : النداء max (a, b) في المسألة الأولى من الفقرة 7.2.2 (المثل A1) يمكن أن يُعتبر معادلاً لـ :

```
Begin
    INT x is a;
    INT y is b;
    If x > y then result x
            else result y
            end of if
end.
```

جرى إدخال الكلمة mod (modifiable) التي تعني إمكانية التعديل لتجميع المتحولات من النوع ... Var . هكذا ، فإذا كنا نرغب حقاً بأن نجعل متحولة معينة ، متغيرة في إجراء معيَّن (ليس فقط القيمة الأخيرة المخصصة) ، فهذا يعني إن جسم هذا الاجراء يمكن أن يُخصِّص قيمة أخرى إلى هذه المتحولة . هذه هي الحالة في المثل A2 ، فالإجراء change i, j يُمكن أن يُخصِّص قيمة معينة إلى المتحولة Nb : هذه ليست فعلاً الحالة بالنسبة للمتحولات b, a حيث المتحولات الشكلية هي محدَّدة مسبقاً بواسطة fixe .

عندما يكون أحد المتغيرات الوسيطية الشكلية (formal parametre) مثبَّت مسبقاً بواسطة fixe ، وإذا قمنا بإعطائه وتخصيصه بإحدى المتحولات وذلك كمتغيِّر وسيطي فعلي ، فإن قيمة هذه المتحولة هي التي سيتم نقلها إلى داخل الإجراء عند ندائه ولا يتم نقل المتحولة نفسها إلى داخله .

7.2.4 ـ الاجراءات في لغة باسكال

نجد في لغة باسكال نوعان من الاجراءات :

الإجراء « دالة » (function) ، والاجراء « إجراء » (procedure) .

7.2.4.1 ـ الاجراء (دالة) «function»

الدوال (functions) هي عبارة عن تعميم بسيط للدول النموذجية (Standard function) . بإمكان المستعمل أيضاً أن يُعرِّف ما يستطيع من هذه الدول حسب حاجته .

نداء « الدالة » يتم بالمراجعة في داخل تعبير جبري ، كما هو الحال بالنسبة للدول النموذجية .

```
FUNCTION TG (X: REAL) : REAL;
    BEGIN
        TG: = SIN (X)/Cos (X)
    END;
```

السطر الأول ، ويدعى « رأس » الدالة . النوع الأخير (REAL في هذه الحالة) من نفس السطر ، يُعرِّف على نوع الدالة ، أي نوع القيمة التي ستدخل في كل تعبير يقوم باستعمال الدالة أو نوع القيمة التي ستعود إلى البرنامج الأساسي . بين الأهلة ، يوجد ما ندعوه لائحة بأسماء المتحولات والمتغيرات المستعملة في الإجراء « دالة » . في هذا المثل لا يوجد سوى متغيِّـر واحد هو X من نوع REAL . يتبع المتغيرات أنواعها ، وهي تلعب دور المتغيرات الشكلية عند حسابة قيمة الدالة function .

هكذا ، فالنحو العام المستعمل لصياغة الدالة يبدو كما يلي :

```
FUNCTION identificator (a1: type, a2 : type...) : TYPE;
    BEGIN
      body of function
      END;
```

- FUNCTION : كلمة مفتاح تدل على الاجراء دالة .
- identificator : إسم الاجراء دالة .
- a1: Type, a2: type , ... : أسماء المتحولات الشكلية وأنواعها .
- TYPE : نوع قيمة الدالة .
- body of function : جسم الدالة أو التعليمات التي تؤلف الدالة .

فلنشِر إلى إن ما يسمى متحولات عامة (global variable) هي تلك المعروفة من قِبل البرنامج المُنادى ، بينما المتحولات المركزية (local variable) فهي تلك المعروفة فقط من قِبل الاجراء نفسه .

7.2.4.2 ـ الاجراءات PROCEDURE

بينما تنتج الدالات قيماً معينة بعد تنفيذها ، فإن الاجراءات تقدم أفعالاً في البرنامج الرئيسي وتساعده على حلّ المسألة الأساسية دون تكرار كتابة أقسام منه .

مثلاً : فلنكتب إجراء procedure يقوم برسم خط مؤلف من تكرار السمة C لعدد N من المرَّات .

```
PROCEDURE TRACE (C: CHAR; N : INTEGER);
    CONST MAX = 100;
  BEGIN
    IF N > MAX THEN N : = MAX;
    FOR I : = 1 TO N DO
        WRITE (C);
    WRITELN
END;
```

الفصل الثامن

بناء المعطيات DATA STRUCTURE

8.1 ـ الرتل (أو الصف)

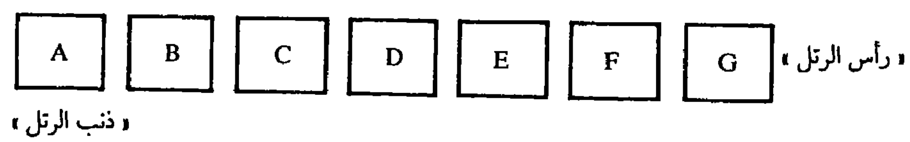
هو عبارة عن مجموعة من المعطيات المختلفة مرتبة حسب ترتيب معين .

يتميز الرتل بعلاقة تراتبية عامة يُرتبط بها عناصره . يُعتبر كل عنصر من الرتل غير قابل للانفصال عن اللائحة ؛ وفي بعض الأحيان ، قد يكون كل عنصر من عناصر الرتل هو نفسه عبارة عن مجموعة أو عن رتل . يُراجع كل عنصر من الرتل أو يتم بلوغه بواسطة رتبته في الرتل ، وبما أن تخزين عناصر الرتل يتم بشكل متقارب في الذاكرة المركزية ، لذلك نستطيع بلوغ كل عنصر منه بواسطة إسم الرتل مزوداً برتبة (rang) العنصر داخل الرتل .

عملية البحث بداخل الرتل تتم بواسطة الكنس المتتالي لجميع عناصره ، لذلك يكفي لبلوغ جميع العناصر أن نقوم بتغيير قيمة المؤشر (index) (المؤشر عبارة عن عدد إيجابي صحيح) وذلك بإعطائه على التوالي جميع القيم المسموح بها .

الوصف المنطقي للرتل :

يبدو الرتل على الشكل التالي :



يمتاز الرتل « برأس » (Head) ، وبذنب (queu) . وحسب طبيعة المعالجة المطلوبة ، فقد نكون بحاجة إلى إجراء عملية فرز جديدة للعناصر وبالتالي تعديل نظام ترتيبها ، كما قد نكون بحاجة إلى إجراء عمليات بحث متكرِّرة داخل الرتل، مما يستدعي إستعمال مؤشر إضافي (pointer) عندما لا نكون ملزمين بالبدء بعملية البحث في أول الرتل

في كل مرَّة مما يستدعي إستعمال المؤشر للدلالة على العنصر الذي سنبدأ منه بالبحث أو بالمعالجة (ذلك للاقتصاد في عملية الوقت بدلاً من البحث إنطلاقاً من بداية الرتل أو من رأسه) .

نستطيع تشبيه الرتل بالمصفوفة ، حيث عناصر المصفوفة هي مرتبة حسب ترتيب معين ، ولكن الفارق يكمن في إمكانية إضافة (insertion) عناصر جديدة إلى الرتل أو إلغاء أخرى أو تعديل نظام ترتيب العناصر .

8.2 ـ الجداول

نعرِّف الجداول ببعدين (two dimensions) وكأنها عبارة عن رتل من العناصر ؛ حيث كل عنصر منها هو بحد ذاته عبارة عن رتل مؤلف من نفس عدد العناصر .

وبالإمكان تعريف الجداول بأبعاد مختلفة ، بثلاثة أبعاد حيث كل عنصر عبارة عن رتل من العناصر ، كل عنصر منها عبارة عن جدول ببعدين . وهناك جداول بأربعة أبعاد حيث كل عنصر عبارة عن رتل من العناصر كلٍ منها عبارة عن جدول بثلاثة أبعاد . . . الخ .

فلنشِر إلى إن بنية الجدول المخزَّن في الذاكرة هي شبيهة ببنية « الرتل » ، لذلك ولمعالجته نحتاج إلى وسيلة تسمح باستغلال الجداول بطريقة شبيهة بتلك التي نلتقيها عند معالجة المصفوفات . من هنا ولبلوغ عنصر من الجدول فنحن بحاجة :

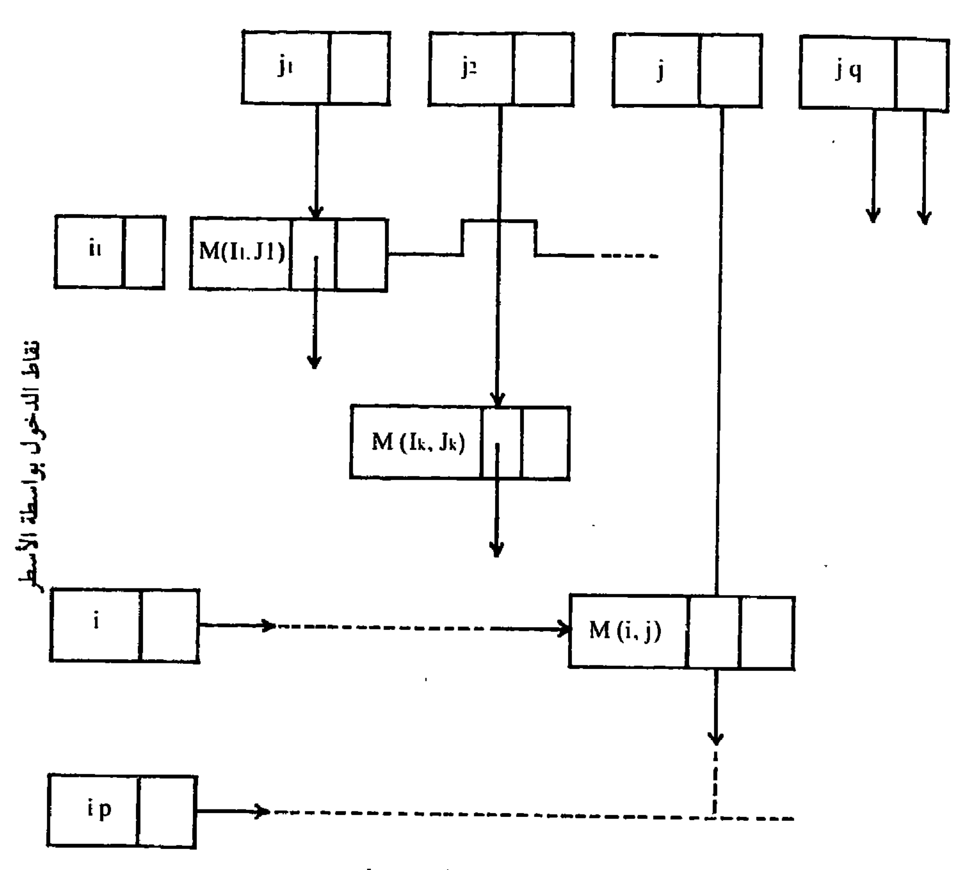
ـ إلى معرِّف عن الجدول .

ـ سلسلة من القيم أو المؤشرات أو الدلائل مرتبطة بأبعاد الجدول .

عملية البحث عن العناصر في الجدول هي شبيهة بتلك المستعملة في « الرتل » ، ولكن ، في الجدول ، بالإمكان تغيير قيمة المؤشرات بشكل مستقل الواحد عن الآخر .

بالإمكان تحويل جدول بعدد n من الأبعاد إلى رتل من الجداول ، بحيث كل جدول هو بعدد $n - 1$ من الأبعاد .

فلنفترض جدولاً M(I, J) ، فباستطاعتنا بلوغ عناصره حسب الشكل التالي :



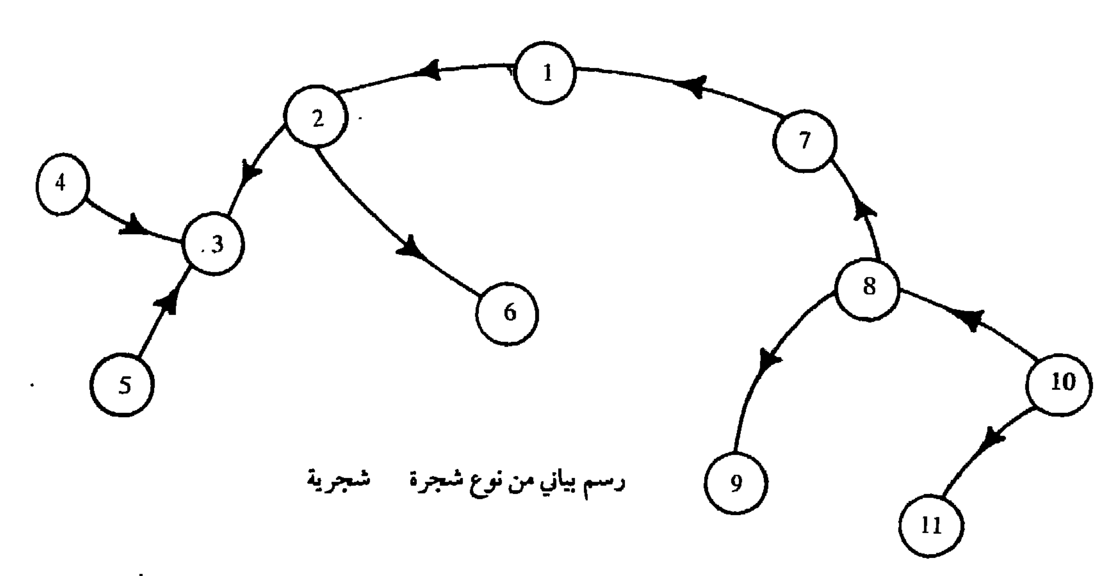
بلوغ العناصر بواسطة الأسطر والأعمدة

بـالإمكان بلوغ كـلّ عنصر من العناصر الجـدول بواسـطة مؤشـرات لـلأعمـدة والأسطر ، ومن داخل الجدول ، وبواسطة العناصر يمكننـا بلوغ عناصر أخـرى داخلية إنطلاقاً من مؤشرات أو مفاتيح تُزوَّد بها العناصر كما نرى في الشكل أعلاه .

8.3 ـ الشجريات (arboresance / tree)

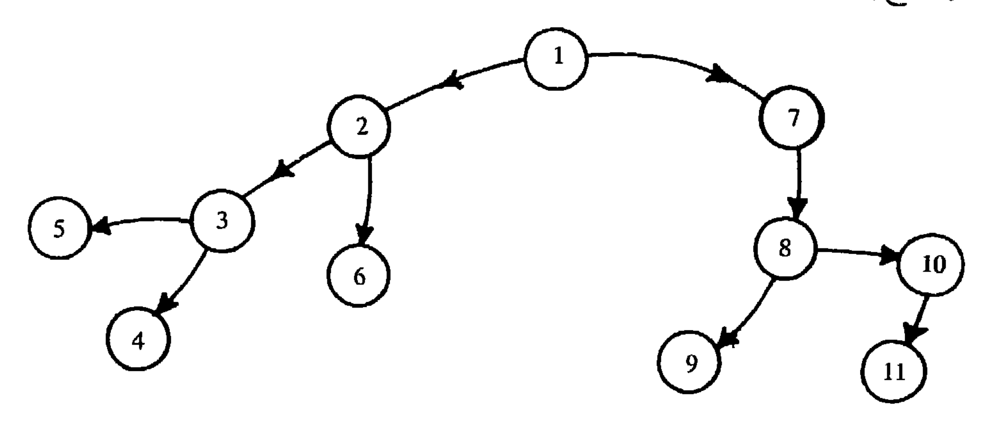
تسمح نظرية الرسوم البيانية بوضع رسوم بيانية غير دائرية تدعى شجريات . تمتاز الشجرة بعدد من القمم والأضلع . وهناك نوعان من القمم :

- قمم بعدة أضلع فرعية ساقطة .
- قمم بضلع واحد ساقط . هذه الأخيرة تدعى معلّـقة

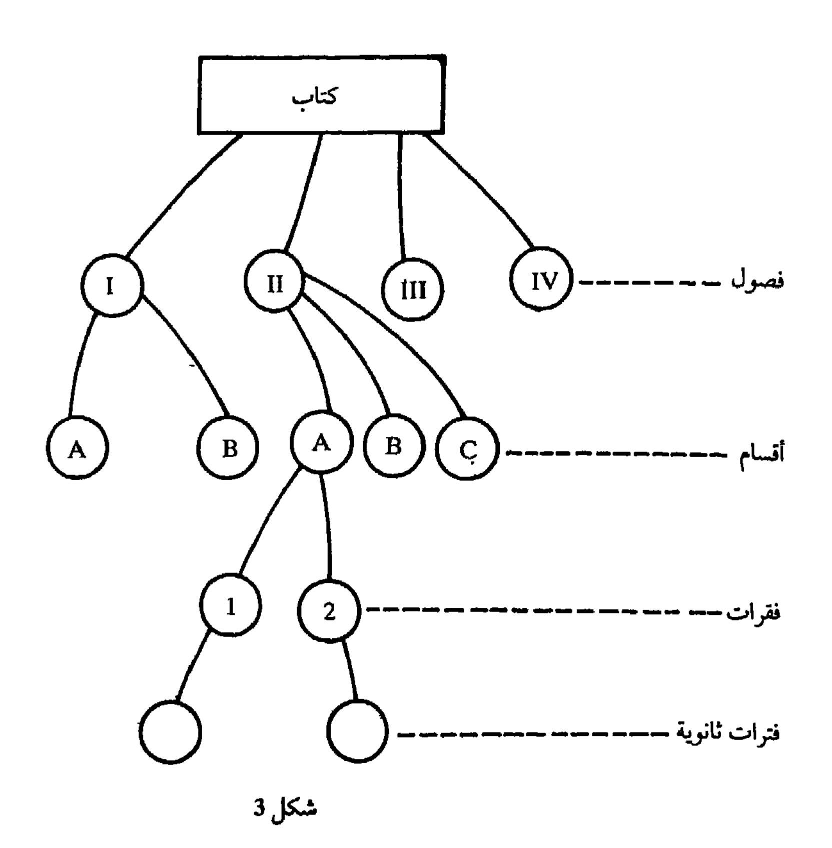


رسم بياني من نوع شجرة   شجرية

الشجرية هي عبارة عن شجرة (tree) حيث كل قمة (ما عدا واحدة تدعى جذع) (root , racine) هي عبارة عن طرف نهائي لقوس واحد . يوجد إذاً في الشجرية قمة (جذع) مرتبط بكل قمة أخرى بواسطة مسلك (طريق) واحد .



تمثل على الشجريات ، نستطيع أن نذكر تنظيم أحد الكتب من فصول ، أقسام ، فقرات ، فقرات ثانوية ، . . . الخ . وذلك على الشكل التالي :



شكل 3

تستطيع القمم المرتبطة بنفس القمة السابقة تشكيل علاقة تعادلية هي : « تملك نفس القمة السابقة » . وفي بعض الأحيان ، تستطيع تـركيبة الشجـرية أن تؤمن عـدة علاقات تراتبية بين العناصر (القمم) . هنا سنشير إلى إثنين منها :

(أ) الترتيب المُسْتَعْرِض :

جميع القيم التي تمتاز بنفس الرتبة ، توضع كما في الشكل 3 على نفس الخط الأفقي . من هنا فمن الممكن إجراء ترتيب جديد لها (من اليسـار إلى اليمين مثـلاً) . هكذا ، فبالإمكان مقارنة قمتين إذا كانوا بنفس الرتبة ، من هنا نستطيع تعـريف علاقـة ترتيب جزئية .

(ب) ترتيب تاري (TARRY)

يُمثّـل الرسم البياني كما في الطريقة السابقة ، وذلك بتسطير القمم التي تمتاز بنفس الرتبة (جعل القمم على سطر واحد) . يقوم نظام ترتيب تاري (TARRY) بالانتقال إنطلاقاً من الجذر (root) ، من قمة إلى قمة يشكل :

1 ـ عند الاتجاه نحو « الأسفل » نأخذ الفرع الموجود في أقصى اليمين والغير مستعار (من اليسار عندما ننظر إلى الرسم) ونتابع ذلك حتى نصل إلى قمة مُعلّقة .

2 ـ من خلال قمة مُعلّقة نتجه « نحو الأعلى » حتى نبلغ قمة بقوس غير مستعار ، فنعود ونتجه إلى الأسفل كما في (1) .

هكذا نستطيع عبور الشجرية كاملةً . وكل قوس سيتم إستعارته (العبور عليه) مرتين . نظام ترتيب القمم هو ذلك الذي نحصل عليه عند ترقيم كل قمة عند أول التقاء بها . هذه هي علاقة تنظيم كاملة .

نقول إن مجموعة من المعطيات تحتوي على تركيبة لوائحية (أو تُمثّل شجرية) إذا كان من الممكن تعريف علاقة تنظيمية بين القمم مشابهة لاحدى العلاقات المذكورة في الفقرات أ و ب .

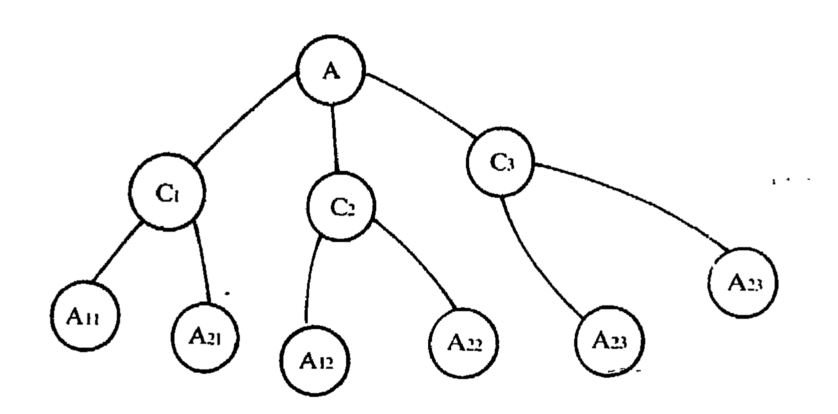
العلاقات المذكورة في الفقرات (أ) و(ب) تسمح بتصوُّر وإعتماد طريقة لتمثيل المعطيات المبنية على أساس الشجرية في الذاكرة بشكل يتأمن معه معالجة سريعة للمعطيات . يتم تزويد كل قمة من القمم بوصلتين للتعليق : الوصلة الأولى تُشير إلى القمة التالية على الخط المُستعرض الأفقي حسب علاقة الترتيب المستعرض ، والوصلة الثانية تشير إلى رتبة القمم التالية .

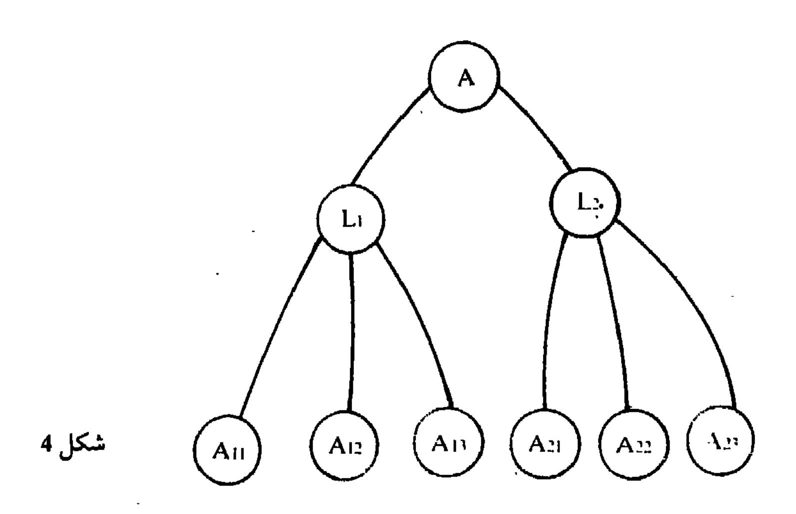
نستطيع فئتين من العمليات على اللوائح : العمليات التي تجري على الشجريات (معالجات إنحدارية أو تصاعدية) ، وتلك التي تحتاج إلى التكرار .

تقوم المعالجات أو العمليات الانحدارية، في الحالة الأسهل ، على عبور الشجرية مع الأخذ بالحسبان للعلاقة التراتبية بين القمم . فقد يكون ضرورياً في بعض الأحيان تخزين القمم التي تبلغها في الحالة التي نجتاز فيها الشجرية نزولاً حسب ترتيب « تاري » . إستعمال مكدس خاص يُسهّل عملية البحث عن الأقواس الغير مُستعارة .

A =

A_{11}	A_{12}	A_{13}
A_{21}	A_{22}	A_{23}



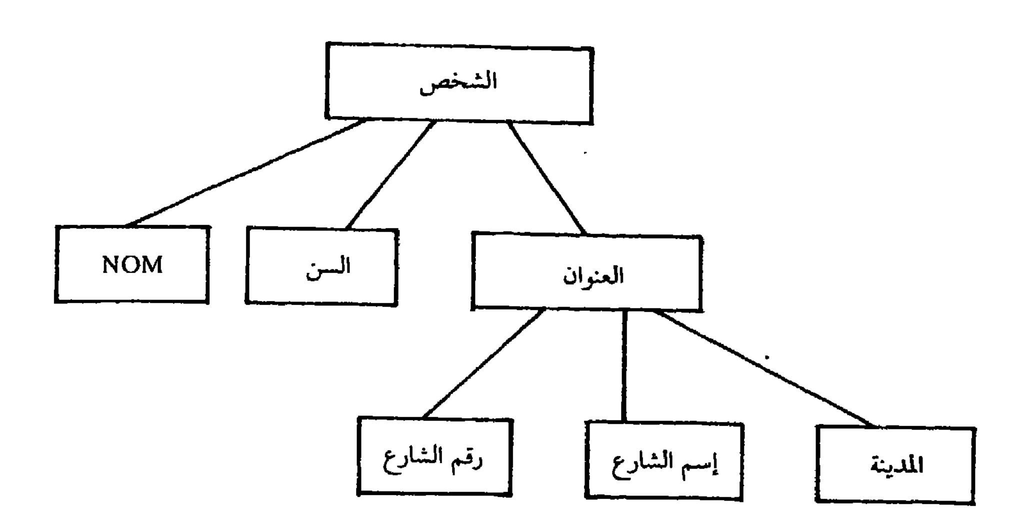


شكل 4

المعالجة التصاعدية هي أكثر صعوبة . نجد هذه التقنية في المصرِّفات التفسيرية (المفسِّرات) عندما يتم تنفيذ البرنامج مباشرة عند إدخاله حسب ترتيب تعليماته . بشكل عام عملية المعالجة تنطلق من الأسفل نحو الأعلى . مثلاً من خلال عنوان الشخص نرغب بمعرفة إسمه .

في أغلب الأحيان ، تستدعي المعالجة التصاعدية إجراء عمليات متكرِّرة ، كما قد تحتاج إلى إستعمال عمليات التكديس .

مثلاً :



باستطاعتنا أن نعبر الشجرية من الأسفل باتجاه الأعلى إنطلاقاً من كل قمة من القمم .

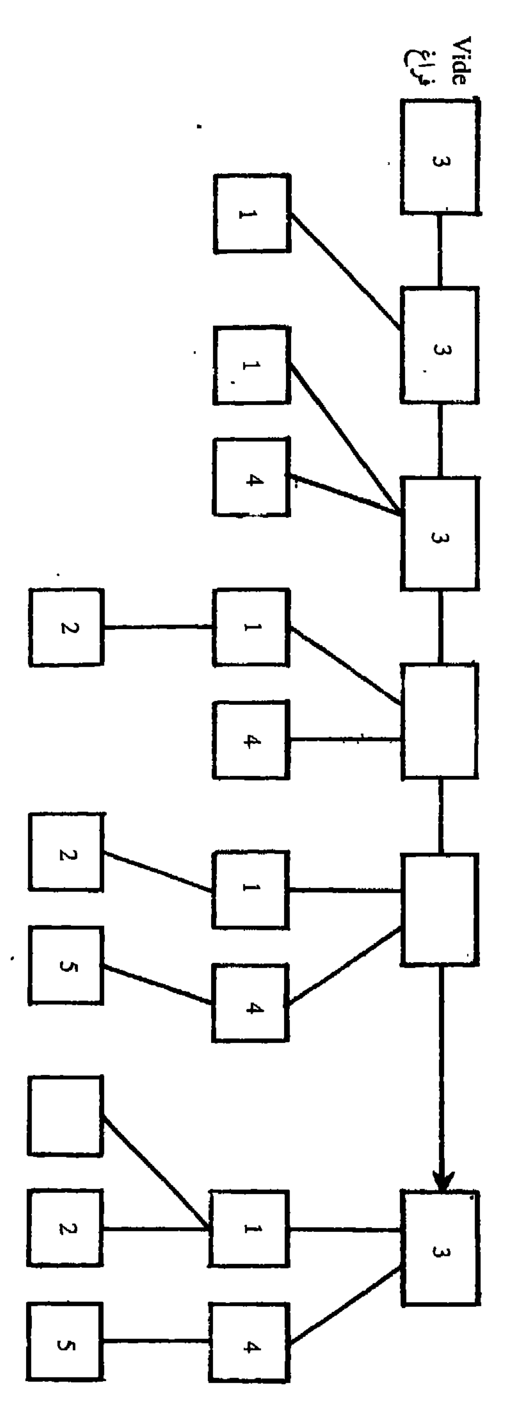
8.4 ـ الشجرة (TREE) والشجرة الثنائية (binary tree)

الشجرة من نوع T هي عبارة عن بنية تتألف من مُعطى من نوع T يُدعى جذع ومن مجموعة مُحدَّدة ، بحجم مُتغيِّر ، وإحتمالاً فارغة ، من الشجيرات من نوع T ، التي تدعى شجيرات ثانوية من الشجرة .

الشجرة الثنائية من نوع T عبارة عن بنية تكون إما فارغة وإما مؤلفة من :

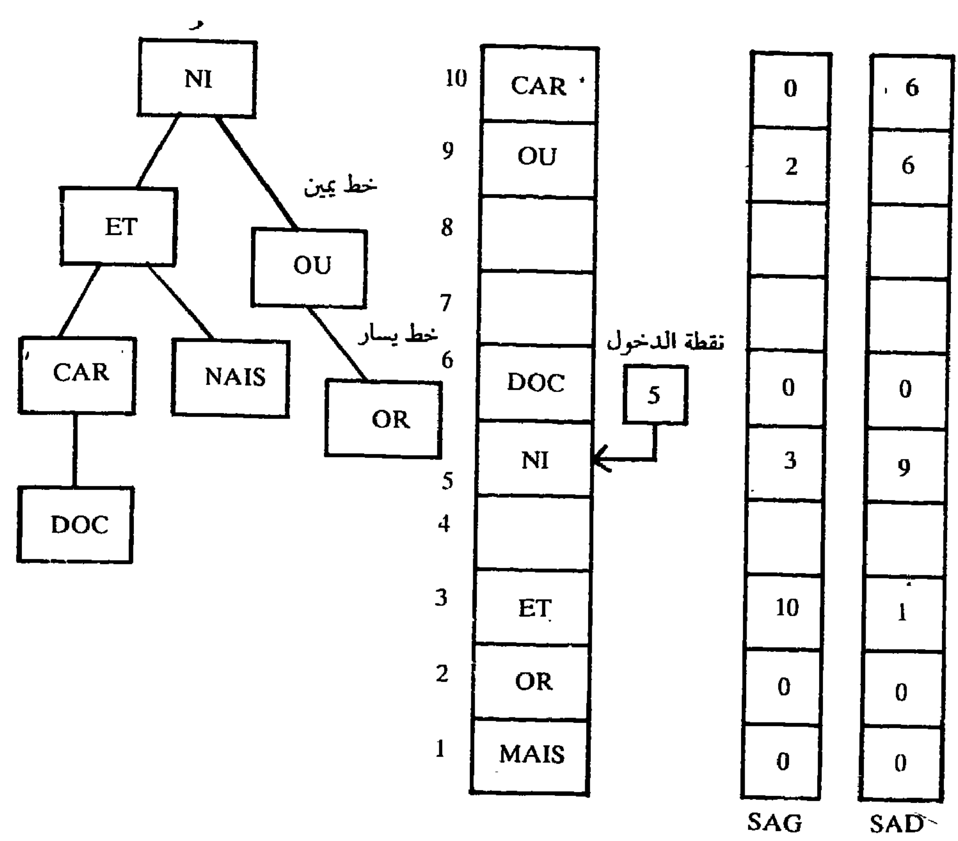
- معطى من نوع T يُدعى جذع الشجرة الثنائية .
- من شجرة ثنائية من نوع T تدعى شجرة ـ ثانوية ، يُسرى (SAG) .
- شجرة ثنائية من نوع T تدعى شجيرة ثانوية ـ يمنى (SAD) للشجرة الثنائية .

بشكل عام ، وعند المعالجة ، نضيف لجدول العناصر جدولين آخرين متوازيين SAG وSAD يحتويان على الوصلات اليسرى واليمنى . الوصلة « صفر » تناسب الفراغ .



مثلاً :

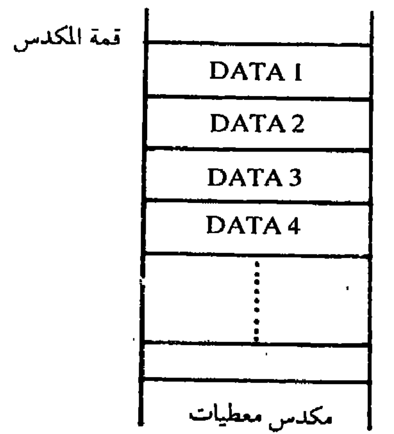
الشجرة التالية هي شجرة ثنائية للبحث إذا كان الترتيب الذي جرى إختياره هو ترتيب أبجدي .



8.5 ـ المكدس (STACK)

المكدس هو عبارة عن مجموعة من المعطيات المرتبة رمزياً الواحد فوق الآخر بشكل نستطيع فيه بلوغ المعطى الموجود في الأعلى فقط.

هذه العناصر المميزة الموجودة في الأعلى تدعى قمة المكدس .



يتطور المكدس خلال المعالجة ، لذا فالعمليات المسموح بها عليه هي التالية :

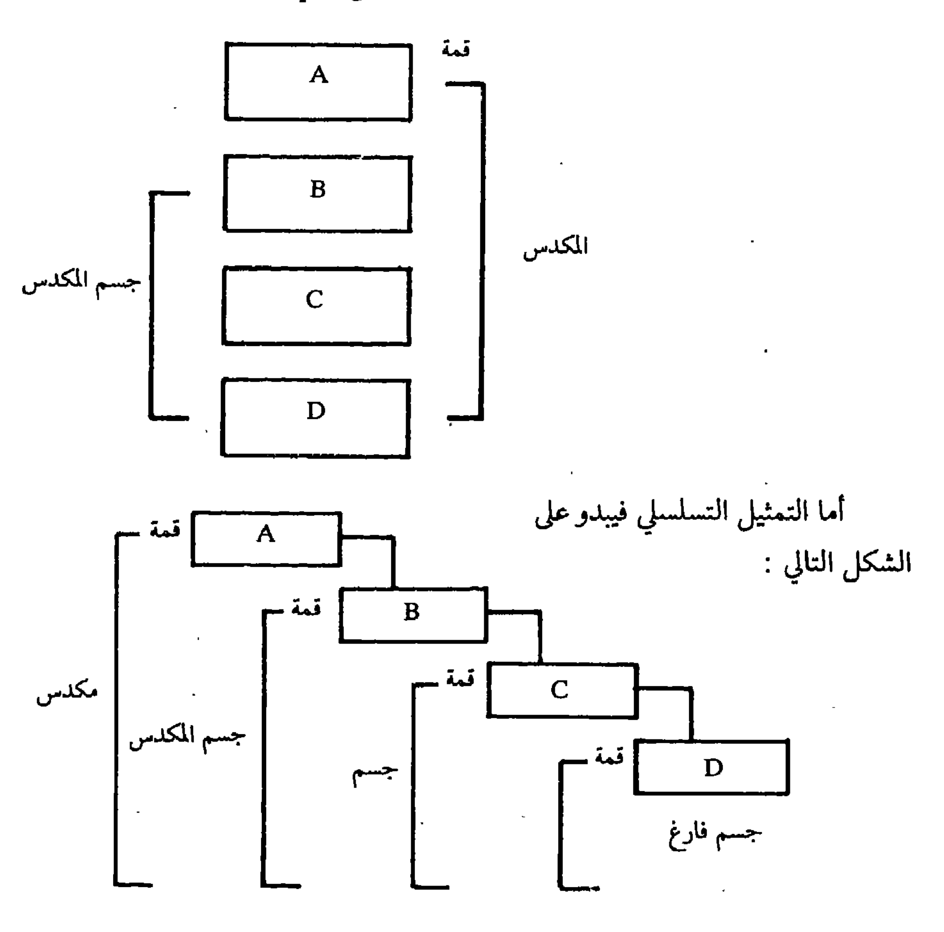
- إضافة عنصر جديد إلى قمة المكدس ، مما يؤدي إلى تخفيف إحتمال بلوغ العناصر الأخرى . نقول بوجود كبس أو رصّ للمكدس .
- إخراج عنصر من المكدس ، مما يؤدي إلى الفعل العكسي أي صعود المكدس .

العنصر الأخير الموجود في المكدس يدعى العنصر الأساسي ، وعندما يخرج العنصر الأساسي من المكدس يصبح هذا الأخير فارغاً .

لمعالجة عناصر المكدس أو المعطيات المخزنة فيه ، يكفي أن نستعمل مؤشرين فقط : مؤشر للدلالـة على القمـة أو العنصر الأول من المكدس ، ومؤشر للدلالـة على العنصر الأساسي أي العنصر الأخير منه . هذا التمثيل المتتالي لا يفرض بالضرورة إستعمال عناوين متتالية لتخزين العناصر في الذاكرة .

إضافة لذلك ، فالتوصيل بواسطة الحلقات يسمح لكل عنصر بالإشارة إلى العنصر التالي (العنصر الموجود في الموقع الأسفل) ، وعندما يؤشر العنصر الأساسي على القمة نحصل على بنية اللائحة التي تسمح بأفضل إستعمالٍ للذاكرة .

هكذا فالتمثيل المتراص للمكدس يبدو على الشكل التالي :



أما التمثيل التسلسلي فيبدو على الشكل التالي :

عند إضافة مؤشر لكل معطى نحصل على التركيبة التسلسلية . يستعمل المؤشر للاشارة إلى العنصر التالي الداخل ضمن المكدس .

8.6 ـ القوائم (tables)

القوائم عبارة عن تركيبة متعاقبة ، حيث العناصر تتناسب بشكل أزواج لكل عنصرين على حدة (وعنصر برتبة مفردة وعنصر برتبة مزدوجة) . العناصر ذات الرتبة المفردة هي المتغيرات الوسيطة أما تلك التي تمتاز برتبة مزدوجة فتمثل القيم .

وإذا كانت المتغيرات معروفة بشكل ضمني فبالإمكان إهمالها وعند ذلك نحصل على سلّم المعطيات . باستطاعتنا أن نقوم بنوعين من المعالجات على القوائم .

- لنفترض وجود المتغير الوسيطي ، ونرغب بالحصول على قيمته .
- لنفترض القيمة ، ونرغب بمعرفة المتغير الوسيطي المناسب .

للإجابة على هذه الأسئلة نستعمل الطرق التالية :

أ ـ الكنس المتتالي البسيط

هذه الطريقة تقوم على مقارنة المتغير المستعمل للبحث وعلى التوالي مع جميع المتغيرات الموجودة في القائمة ، وذلك حسب تسلسل ورودها .

ب ـ الكنس المتتالي مع الأخذ بعين الاعتبار النسبة الوسيطية للظهور أو للبحث عن كل عنصر من القائمة . العناصر التي تتمتع بالاحتمال الأكبر للإستشارة أو للبحث توضع في أعلى اللائحة . وبالتالي فإن عملية البحث ستتم في مدة أقصر .

ج ـ عمليات الكنس تتم باستعمال طرق مختلفة منها طريقة الفرقان(dichrotomie) أي بتقسيم القائمة إلى قسمين وكنس كلٍ منهما على حدة ، فإذا لم نجد العنصر الذي في القسم الأول نبحث عنه في القسم الثاني بعد تقسيمه هو أيضاً إلى قسمين وهكذا دواليك .

8.6.1 ـ القوائم العشيرية أو التراتبية

تستعمل هذه القوائم لتسريع عملية البحث بداخل القائمة . وهي تقوم على إستعمال قائمة إضافية T1 تمثّل قسماً في القائمة الأساسية T .

في القائمة الإضافية T1 ، يجري إضافة مؤشرين لكل متغير :

- المؤشر الأول يُستعمل إذا كان المتغير الذي نبحث عنه أصغر من المتغير الموجود في T1 .

ـ المؤشر الثاني يُستعمل إذا كان متغير البحث يعادل المتغير الموجود في T1 ؛

مثلاً : لنفترض قائمة بالأحرف الأبجدية ونبحث فيها عن أحد الأحرف .

E	1	5
J	6	10
O	11	15
T	16	20
Z	21	24

T1

1	A	
2	B	
3	C	
4	D	
5	E	
6	F	
7	G	
8	H	
9	I	
10	J	
11	K	
12	L	
13	M	
22	V	
23	W	
24	X	
25	Y	
26	Z	

في البداية يجري البحث داخل القائمة T1 . وإذا كان مؤشر العنصر الذي نبحث عنه أصغر من المؤشر الثاني ، نقوم بمقارنته بالمؤشر الأول لتحديد الحيز الموجود فيه هذا العنصر كما نلاحظ في الشكل أعلاه . بواسطة هذه الطريقة سيكون باستطاعتنا تسريع عملية البحث بشكل كبير .

8.7 ـ تنظيم نظام للمعلومات بواسطة الأهلّة والتمثيل الشجري للمعطيات

8.7.1 ـ مقدمة :

يتعلّق ذلك بأحد المواضيع في عالم المعلومات ـ تمثيل المعطيات بشكل مريح لإنشاء نظام للمعلومات . لإدخال وإخراج المعطيات المطلوبة ، مع تخفيض لمدة البلوغ .

فمن المعلوم إن أحد أهم مميزات هذا العنصر هو المعلومات ، حيث تضاعفت كمية المعطيات الفعالة في كل يوم وخاصة في العشر سنوات الأخيرة . وتتلخص أزمة المعلومات الظاهرة في بعض البلدان بنتيجة ضياع كمية كبرى منها بسبب سوء تنظيم المعطيات في المعطيات .

الـواسطة الفعـالة لتفـادي الأزمات في ضيـاع المعلومات تبـدو الآلات الحـاسبـة الإلكترونية .

فقد لاقت المعلومات ونـظم المعلومات رواجـاً منقطع النـظير في جميع المجـالات الانتقادية (التجارة ، البنوك ، الشرطة ، المزارع ، الادارة ، المستشفيات الخ) ، كذلك في مجال العلوم التقنية . حيث تلعب نظم المعلومات دوراً مهماً في إدخال وإخراج المعطيات المفيدة . وفي السنوات الأخيرة صنع العديد من الشركات أنظمة للمعلومات مُتخصّصة . مثلاً : النظام IBM5520 المستعمل في الإدارة ولصناعة الوثائق ، النظام ECOM ، PAK من شركة ITT ، والشركة WANGصنعت نظام لمعالجة المعلومات إستعملته على مكناتها ، الشركة HP صنعت النظام Imag ، QUERY لتخزين المعطيات وإنشاء بنوك المعطيات ، ومن ثم صنعت IBM النظام SQL لنفس الغرض . . . . الخ .

غالبية هذه الأنظمة تستعمل الآلات الحاسبة الكبيرة المُجهّـزة بذاكرة خارجية بحجم كبير لتخزين بنوك المعطيات هذه . ويتلخص دور الآلات الحـاسبة ليس فقط لتخـزين المعطيات ولكن لإدارة وتنظيم عمليات بلوغ الزبائن للمعطيات وبالتالي قراءة وكتابة أو إدخال وإخراج المعطيات المختلفة في أي لحظة .

وبالإضافة إلى الأنظمة الكبيرة ، فهناك أنظمة مُتخصّصة صغيرة يصنعها المُستعمِل لاستعمالها في مجالٍ تطبيقي معين ، مثلاً : في الشرطة ، في الشركات الصغيرة الخ . . . في هذه الأنظمة ، هناك مشترك واحد يعمل عليها في نفس الوقت ، وبالتالي لا يوجد تنازع لبلوغ المعطيات ، وبالتالي فإن بساطـة العمل تسمـح بتسهيل عمليـة تنظيم المعلومـات وتخزينها .

من المسائل المهمّـة في عملية تنظيم المعلومات هو خوارزم بلوغ المعطيات وخوارزم الإدخال والإخراج ، بهذا الخوارزم تتعلّـق فعالية النظام العامة . من هنا فإن موضوعنا هو كيفية تنظيم نظام المعلومات كي نحصل على استغلال واضح وأسهل للنظام دون حدوث أية زيادة في نسبة الوقت لبلوغ المعطيات أو لإدخالها وإخراجها .

8.7.2 ـ توزيع المعلومات حسب مميزاتها

فلنفترض إن مجموعة المعلومات عبارة عن مجموعة من العناصر (المميزات) ، التي تُمَيِّـز نوعاً معيناً من المعطيات ، مثلاً ، مميزات مالية ، مميزات تجارية ، . . .

من هنا ، فمجموعات المعلومات A وB (شكل 2) ، أو أنواع المعلومات ، قد تتمتع بعناصر مشتركة . مثلاً: الزجاج هو عبارة عن مجموعة من المميزات : سرعة ، نوع ، سعر ، . . . الخ .

تحتوي مجموعة المعلومات على عدة مجموعات ثانوية من المميزات أو على مجموعات ثانوية من المعطيات . مثلاً :

ـ يحتوي الكومبيوتر على مجموعات ثانوية من العناصر (مُعالج مركزي ، أقراص ، أشرطة مغناطيسية ، . . .) ، كل عنصر منها يحتوي على مميزاته الخاصة .

ـ مجموعة من الأوراق النقدية تحتوي على مجموعة من الوحدات النقدية (دولار ـ سنت ، ليرة ـ قرش . . .) .

توزع المميزات حسب أولويتها ، وهي ضرورية لوضع بنك المعلومات .

8.7.2.1 ـ العمليات الجارية على مجموعات المعلومات

ـ التناسب بين مجموعتين A وB ، إذاً لكل عنصر من A يناسب عنصر من المجموعة B (شكل 3) .

$$\bigcap_{n=1}^{\infty} A_n = \bigcap_{n=1}^{\infty} B_n$$

ـ تناسب بسيط بين A وB ، إذا كان لكل عنصر من A يناسبه عنصر شبيه من B .

$$A \cap B = m$$

ـ إذا كانت المجموعة C مؤلفة من عدة مجموعات A ، B ، . . . ، فمميزات المجموعة C نحصل عليها من خلال مميزات A وB . . . (شكل 4) .

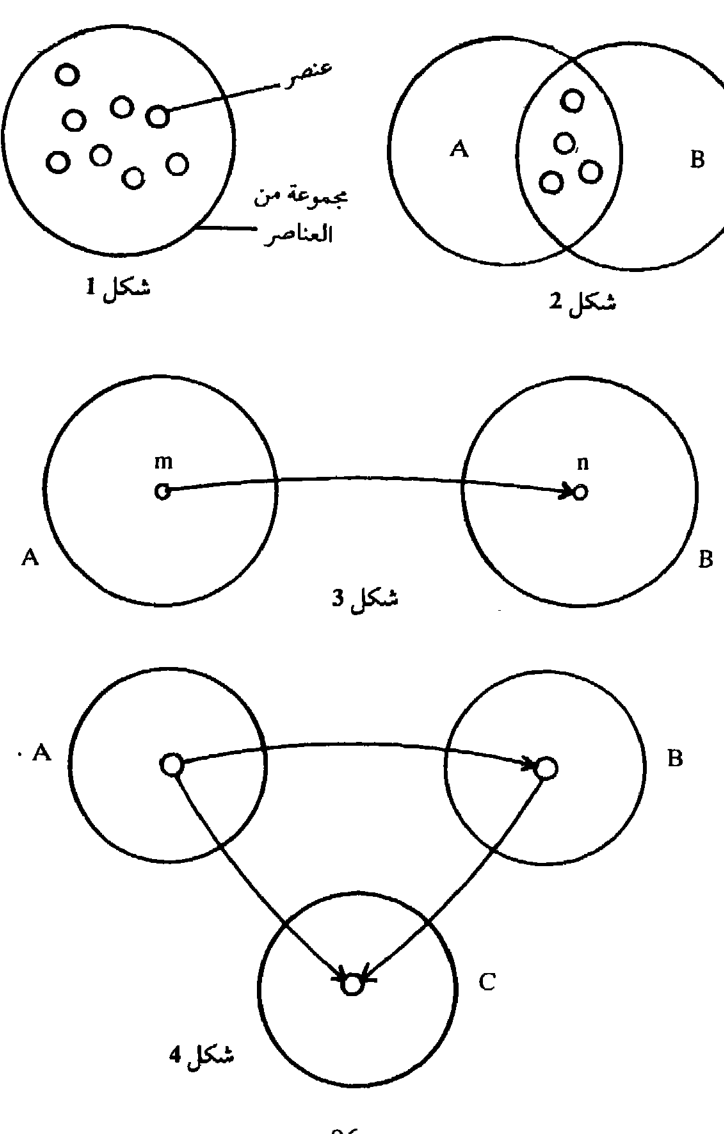
$$\bigcap_{n=1}^{\infty} A \bigcap_{n=1}^{\infty} B \cup \ldots = \bigcap_{n=1}^{\infty} C$$

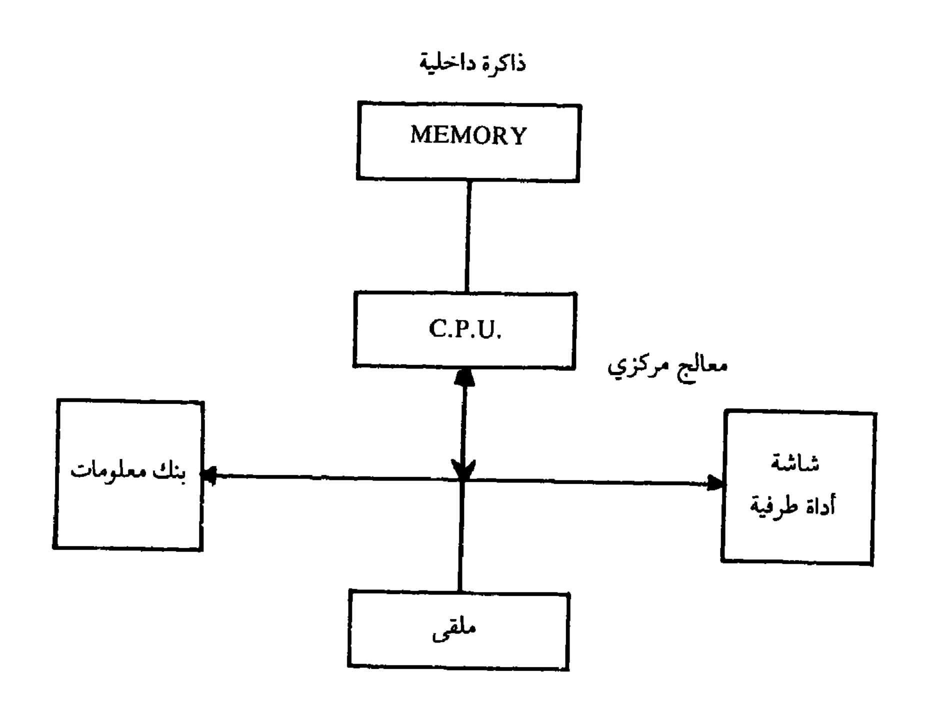
بإمكاننا إستعمال هذه الصفات لبنـاء قدرة من المعلومـات ، أو لتحديـد مميزات مجموعة من المعلومات ، التي تتألف من عدة مجموعات معلومات ثانوية . مثلاً : لبنـاء

إحدى المكنات يجب إستعمال عناصر من عدة أنواع ، عند ذلك تكون مميزة المكنة تعادل مجموع مميزات العناصر التي تتألف منها هذه المكنة .

من الممكن إستعمال نفس هذه المسألة لوصف سلعة معينة أو في عالم الطب . مثلاً : الحرارة + السعلة + ... = مرض الكريب ، وهذا المرض يُعالج بمعالجة مسببات الحرارة ، السعلة وفيروس الكريب . . .

على الرسم 5 ، يوجد مخطط لنظام من المعلومات .





شكل 5

المصاعب التي تظهر عند بناء نظام للمعلومات هي :

1 ـ بناء مجمع للمعطيات (DATA BASE) ، هو لقراءة وكتابة (تخزين) المعلومات . للقيام بذلك يلزم إستعمال ذاكرة بحجم كبير (أقراص مغناطيسية) .

2 ـ إيجاد الطريقة الرياضية لبناء مجمع المعطيات ، هذه الطريقة يجب أن تؤمن سرعة بلوغ كبيرة ، وتؤمن عدم الوقوع في التنازع على المعطيات عند الحاجة إلى بلوغ المجمع .

3 ـ إيجاد الطريقة الرياضية المناسبة لوضع خوارزم التعرُّف على المعطيات ، وقراءة وتخزين المعطيات من الذاكرة الخارجية .

8.7.3 ـ بناء بنك المعطيات

عند بناء بنك للمعلومات تواجهنا بعض الصعوبات المطلوب حلَّها .

1 ـ مسألة التأويل ، تمثيل المعطيات بلغة صريحة لاستخراج المعلومات ، وبإمكانية إجراء العمليات عليها .

2 ـ مسألة التشكيل : تتعلّق بإدخال وإخراج المعلومات المطلوبة ، تنفيذ التعليمات

الواردة بالنسبة للمعطيات ، وترجمة ذلك بالتمثيل الداخلي للمعلومات على الذاكرة الداخلية أو الخارجية .

3 ـ مسألة إدارة نظام المعلومات ، تتعلّق هذه المسألة بالمشاكل الناتجة عن إدارة النـظام بكامله ، وتنظيم عمليات البلوغ الى المعطيات وحمايتها ، وعمليات تمثيل المعطيات بكاملها عند التخزين .

4 ـ مسألة التنفيذ ، ويدخل فيها عملية مزامنة الأعمال ، حماية الذاكرة ، معالجة الحالات الطارئة .

5 ـ مسألة إخراج المعلومات المطلوبة (تشكيل المعطيات ، وتحويلها إلى شكل لائق ومريح للعمل والمعالجة ، إخراج المعلومات على الطابعة أو على الشاشة) .

7.4 ـ إدخال المعطيات

عند إدخال المعطيات ، يتم تجزئة كل مجموعة إلى مجموعات ثانوية . تتـألف كل مجموعة من عدد من العناصر (شكل 6) ، كما وتتألف العناصر من مجموعة من المميزات التي تُميِّـز العناصر من جهة ومن جهة أخرى تعرِّف جميع المميزات عن المجموعة الكاملة .

تتابع عملية التقسيم حتى نصل الى مميزات أساسية للمعلومات غير قابلة للتجزئة وبالتالي فهي ذات دلالة أساسية .

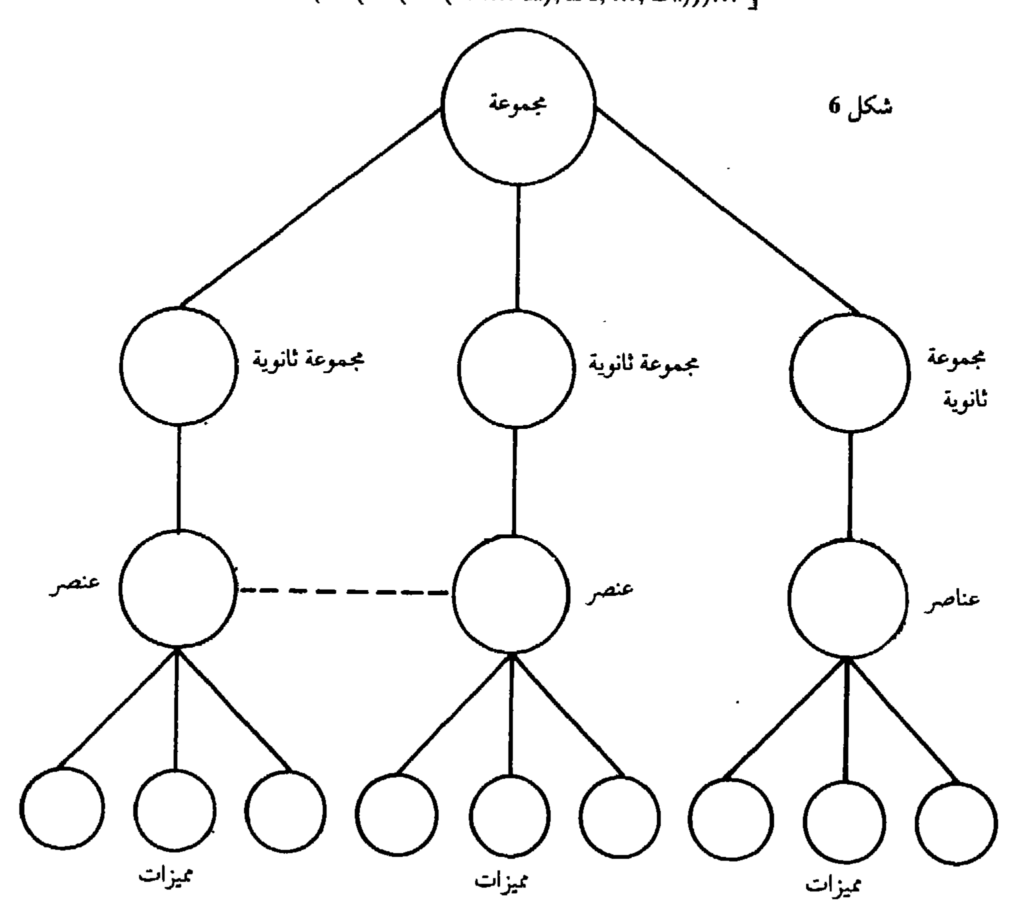
مثلاً : يتألف الكومبيوتر من مُعالج مركزي ، نظام للتشغيل . . إلى ما هنالك ، بينما تتألف المجموعة الثانوية مُعالج مركزي من وحدة العمليات المنطقية والجبرية A.L.V من ذاكرة ثابتة ROM ، من أقنية ووصلات . . . إضافة إلى ذلك فوحدة العمليات المنطقية والجبرية تتألف من دارات تحتوي على رجرجات (trigger) وعلى دارات تكاملية (IC) ، . . . الخ . تمتاز الرجراجات والدارات التكـاملية بسـرعتها (speed) ، بسعـرها . . . بحجمها . . جميع هذه المميزات تُعبِّـر عن المراصف (registre) ومن هناك تعبِّـر جزئياً عن مميزات وحدات العمليات المنطقية والجبرية .

عند ترتيب المعلومات يمكن أن نحصل على مميزات مشتركة ، تُعبِّـر عن مجموعتين ـ ثانويتين مختلفتين من المعلومات . وفي أغلب الأحيان تُستعمل هذه المميـزات المشتركـة كمفاتيح لبلوغ المعطيات ، كما وتأخذ بعين الإعتبار عن التخزين لكي لا يحـدث هناك إسهاباً في المعلومات ، وبالتالي خسارة في حجم الذاكرة المشغول .

عملية تجزئة المعلومات حسب مميزاتها تتابع حتى نصل إلى مميزات خاصة تُعبِّـر عن مجموعة ثانوية واحدة فقط .

بالإمكان تمثيل هذه العملية على الشكل التالي :

$$M\ [\ A_1\ (B_1\ (C_1\ (D_1\ (d_1 \ldots d_n),\ D_2\ (\ldots),\ \ldots,\ D_n)))\ ]\ \ldots$$
$$A_2\ (B_2\ (C_2\ (\acute{D}_1\ (d_1 \ldots d_n),\ \acute{D}_2,\ \ldots,\ \acute{D}_n)))\ldots\ ]$$

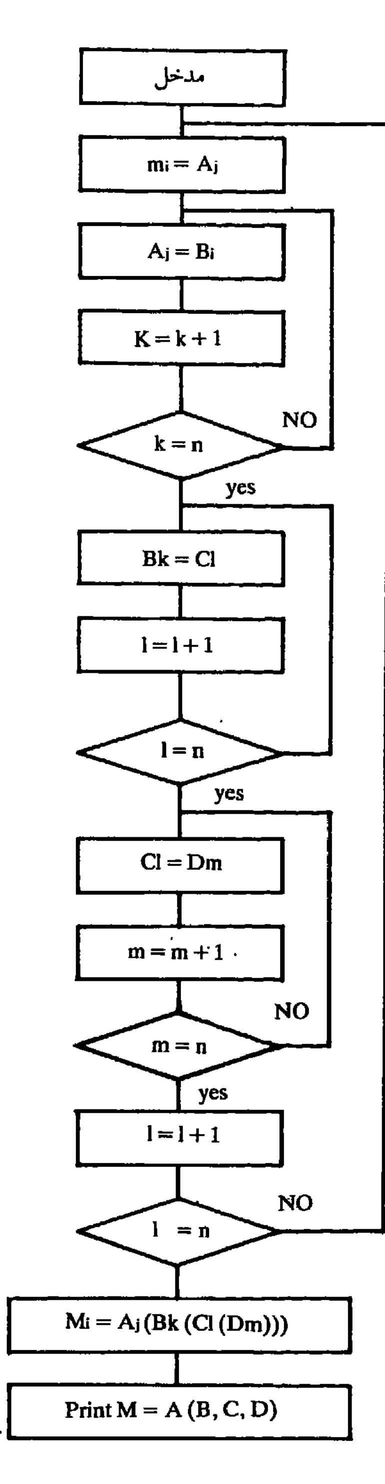


شكل 6

من جهة أخرى يجب أن نُعير إنتباهنا إلى حجم الذاكرة الموضوع بتصرفنا لتخزين المعلومات . عملية إدخال المعلومات وإخراجها تتم حسب الطريقة المتبعة في المعاجم للبحث عن الكلمات ، أي بإدخال عنوان الفقرة نحصل على المعلومات الكاملة الواقعة تحت هذا العنوان .

خوارزم الإدخال

الشكل 7 ، يرسم خوارزم إدخال المعلومات حسب طريقة الأهلّة والتوزيع حسب المميزات .


مدخل
mi = Aj
Aj = Bi
K = k + 1
NO
k = n
yes
Bk = Cl
l = l + 1
l = n
yes
Cl = Dm
m = m + 1
NO
m = n
yes
l = l + 1
NO
l = n
Mi = Aj (Bk (Cl (Dm)))
Print M = A (B, C, D)

شكل 7

خوارزم الاخراج

نبحث عن المجموعة B بالمميزات d_1 وd_2 .

1 ـ يتم تكويد أو تمثيل B ، d_1 ، d_2 بنفس الطريقة التي جرى بها إدخالها في الذاكرة . بعد ذلك يجري البحث عن المعطيات التي تمتاز بنفس المميزات d_1 ، d_2 (حسب طريقة المعاجم الإلكترونية) .

يتم البحث حسب الميزة d_1 وبعد ذلك يجري البحث عن الميزة d_2 وفي النهاية يجري البحث عن المجموعة B (بالمميزات d_1 ، d_2) . يتم تجميع المعلومات التي تمتاز بالميزات d_1 وd_2 وذلك في B : $B(d_1, d_2)$ ، بعد ذلك يجري إخراج المعلومات على الطابعة . مثلاً . نفترض بأننا نرغب بالبحث عن المعلومات الخاصة بإحدى الدارات (رجراج مثلاً) حسب المميزات التالية : النوع K والسرعة 1ns . قد نحصل على عدة دارات بذات المميزات ، لذلك يجب أن يتم إدخال إسم المجموعة (مثلاً رجراج) ، فعند ذلك ستختص المعلومات بالدارات الإلكترونية التي تُمثّل رجراجات من نوع K وبسرعة 1ns .

تُمثّل رجراجات من نوع K وبسرعة 1ns .

مسألة :

مثلاً : لنفترض نظام المعلومات المستعمل في خدمة إحدى شركات الطيران (شكل 9) حسب طريقة الشجيرات .

نرغب بمعرفة الرحلات الى إحدى المدن وتوقيت كل رحلة . في هذه الحالة ستكون المدينة هي أساس (جذع) الشجرة ، وهي ستكون عبارة عن دالة تتأثر بالزمن ، برقم الرحلة ، إسم الشركة ، . . . الخ .

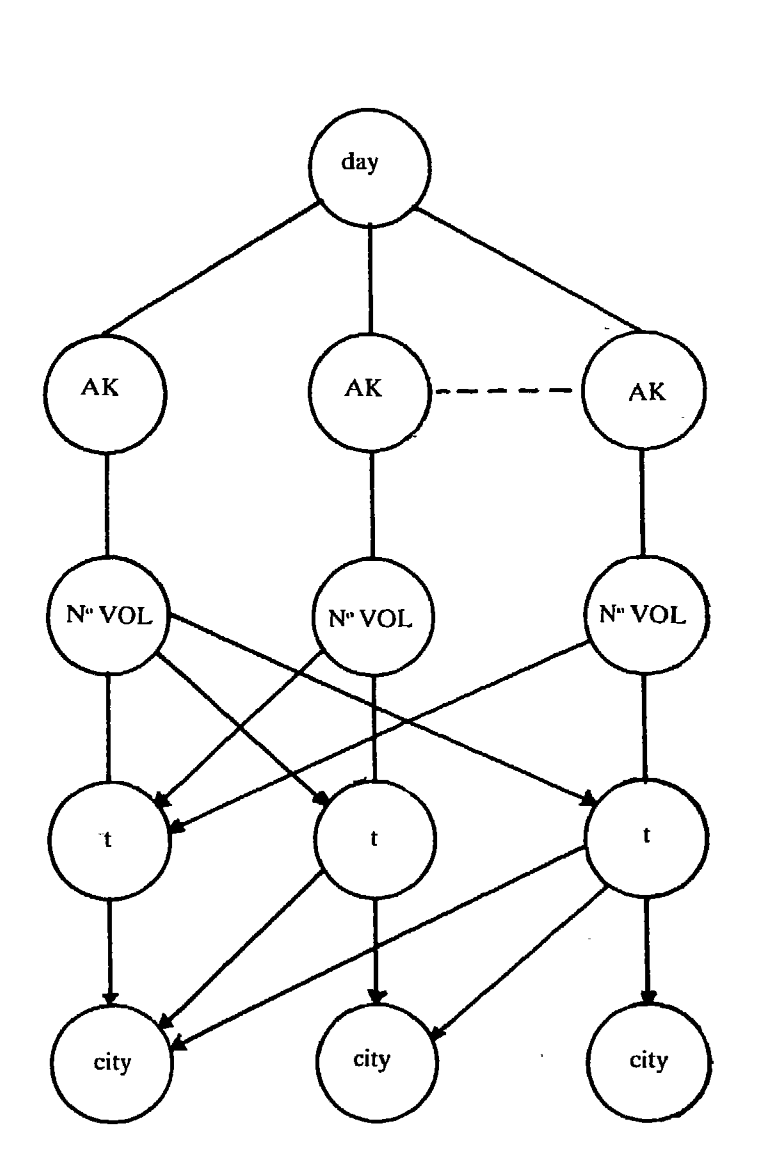
$$city\ [\ t_1\,(AK_1, AK_2 \ldots), \ldots t_2\,(AK_1, \ldots AK_n) \ldots\ ]$$

أو

$$city\ [\ t_i\,(AK_i, N^{o}\ VOL)\ ]$$

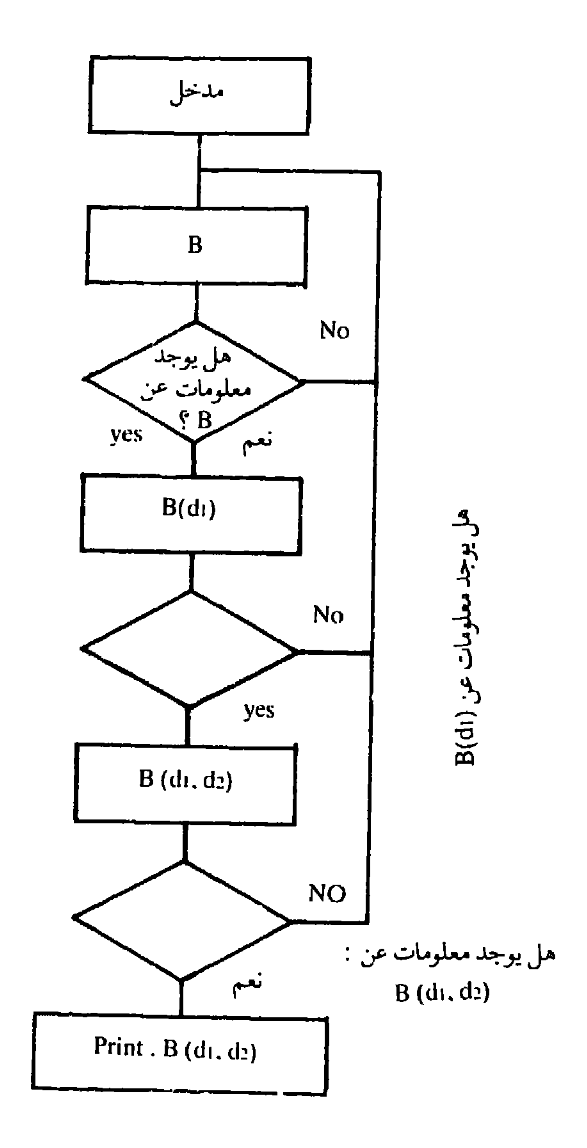
وعلى العكس ، نرغب بمعرفة الرحلات التي تقوم بها إحدى شركات الطيران (AK_i) إلى جميع المدن ، هذه الرحلات ستكون عبارة عن دالة بمتغيرات عبارة عن الزمن t والمدن (city) :

$$AK\ [\ t_i\,(city\ i)\ ]$$



t ـ عبارة عن التوقيت .
city ـ المدينة .
N° VOL ـ رقم الرحلة .
AK ـ شركة الطيران .

شكل 9


مدخل
B
هل يوجد معلومات عن B ؟
No
yes
نعم
B(d1)
هل يوجد معلومات عن B(d1)
No
yes
B (d1، d2)
NO
هل يوجد معلومات عن :
B (d1، d2)
نعم
Print . B (d1، d2)

شكل 8

ـ المعلومات الخاصة بأحد الشركات يمكن أن يتم تنظيمها بنفس الطريقة : (شكل 10) .
هنا سيتم تنظيم المعلومات في ثلاثة مستويات ، والمستوى الثالث والأخير سيكون موحد وثابت بالنسبة للمستوى الثاني :

F [Dep (EMPL (adresse, salary, profession, stage ...))]

EMPL [F (Dep (profession, salary, adresse))]

F ـ شركة .
EMPL ـ عامل .
DEP ـ القسم .
Proffession ـ المهنة .
adresse ـ عنوان
salary ـ المعاش .

المسألة المهمة عند البحث عن المعلومات هي بساطة وسهولة عملية البحث إضافة إلى سرعة البحث.وعدم حصول أي تنازع عن المعلومات ، إذا كان هناك عدة مُستعملين يعملون في نفس الوقت على نفس بنك المعلومات . مثلاً : مراكز الشرطة في جميع أحياء المدينة تعمل على نفس المعلومات ، وبالتالي قد يحدث أن يطلب أكثر من مركز واحد بلوغ بنك المعلومات في نفس الوقت . هذه المسائل تقع على عاتق نظام إدارة بنك المعلومات .

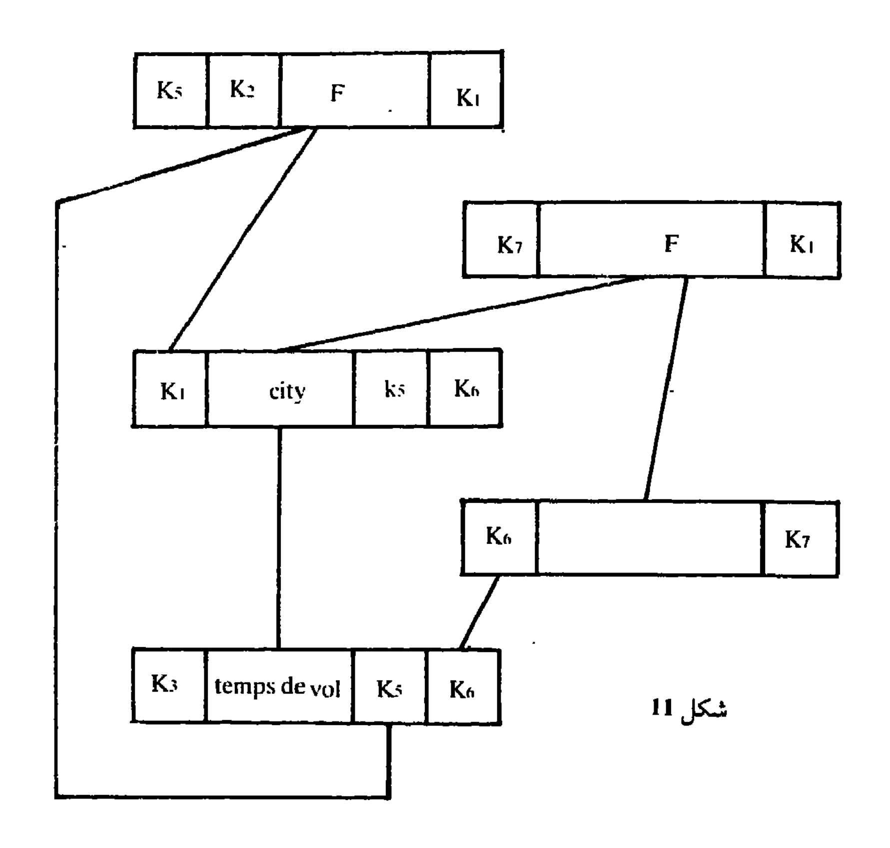
وبشكل عام ، فإن البحث عن المعلومات يتم على عدة مراحل ، وذلك حسب عدد قمم الشجرة وفروعها ، أو حسب عمق الأهلة بداخل التعبير التراتبي للمعلومات .

مثلاً :

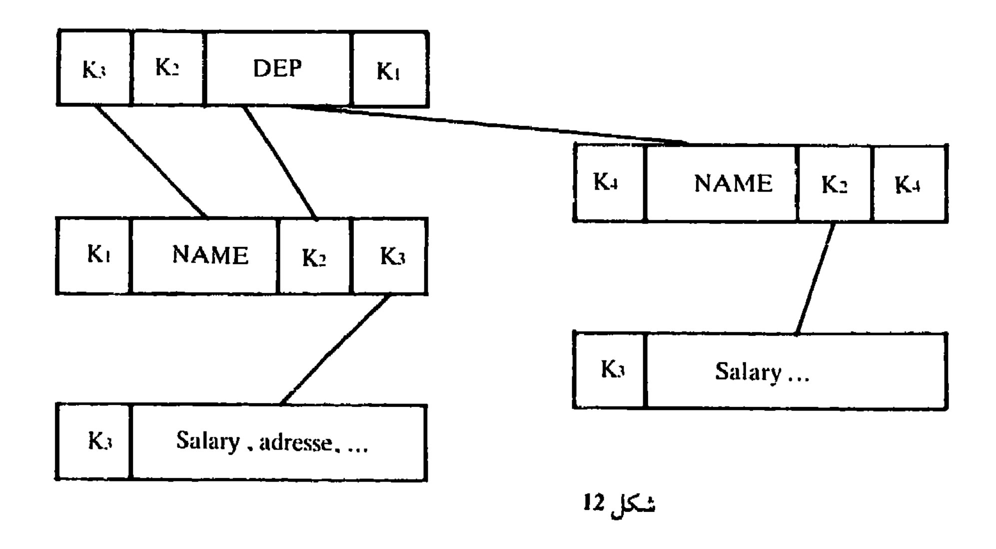
نرغب بالبحث عن رحلات شركة MEA في أحد الأيام ، في البداية يجب أن نبحث عن الشركة MEA ، وبعد ذلك عن المدن التي تصلها هذه الشركة . يجب تزويد كل معلومة بعدد من المفاتيح التي تؤمن بلوغ المعلومة التالية المطلوبة .

والعامل على نظام المعلومات ، سيهتم فقط بالمفاتيح التي تظهر أمامه على الشاشة ، والتي عند إدخالها يحصل على المعلومة التالية المطلوبة (1) شكل (12, 11) .




K5
K2
F
K1
K7
F
K1
K1
city
k5
K6
K6
K7
K3
temps de vol
K5
K6

شكل 11


K3
K2
DEP
K1
K4
NAME
K2
K4
K1
NAME
K2
K3
K3
Salary ...
K3
Salary , adresse, ...

شكل 12



الفصل التاسع

خوارزميات البحث والفرز

9.1 ـ البحث عن عنصر بداخل جدول

لنفترض جدولاً يدعى L ، ويتألف من N من العناصر المنظمة بحيث إن :

$$L_{(i)} < L_{(i+1)}, i = 1, 2, \ldots, N-1$$

لنفترض قيمة معينة A . المسألة تقوم على :

ـ البحث عن القيمة A في داخل الجدول ، وإيجاد ومعرفة المؤشر i ؛ بحيث إذا كانت القيمة A في الجدول نحصل على $A = L_{(i)}$.

ـ إذا لم تكن القيمة A في الجدول ، فالمطلوب البحث عن مؤشر العنصر الذي تسبق قيمته مباشرة القيمة A (أي إذا كان $A < L_{(2)}$ ، يكون المؤشر المطلوب هو 1) .

9.1.1 ـ البحث المتتالي (Sequential research)

الخوارزم الأسهل يقوم على العبور المتتالي للجدول L ، بواسطة حلقة مزودة بمؤشر : $L_{(i)} \leqslant A < L_{(N)}$ ، وإختبار موقع توقف الحلقة أي عندما نجد العنصر الأكبر من A ، أي : $A < L_{(i+1)}$.

نلاحظ إن الحالات $A \geqslant L_{(N)}$ و $A < L_{(1)}$ لا تتوافق مع المؤشر المختار . من هنا نحصل على الخوارزم التالي :

```
RS : if a ≥ L(n) the i : = n
  else if a < L(1) then i : = 0
     else begin i: = 1;
        while a ≥ L (i + 1) DO
            i : = i + 1
          end;
```

```
present : = if i = 0 the false
     else if a = L(i) then true
          else false
```

تقييم الخوارزم

مدة تنفيذ البرنامج R.S تتعلّق بقيمة المؤشر الذي نبحث عنه ind (A) . هناك N + 1 قيمة ممكنة للمؤشر ind (A) . فلنترك جانباً الحالات الخاصة حيث ind (A) = 0 أو ind (A) = N ، ولنأخذ فقط الحالات حيث 1 − 1≤ ind(A) ≤ N ، في هذه الحالات سيتم تنفيذ الحلقة لعدد يعادل ind (A) − 1 من المرتبات . وحسب قيمة A ، فقد يتم تنفيذ الحلقة لعدد يعادل 0، 1، 2 . . . أو N − 2 مرّة .

9.1.2 ـ البحث الفرقاني (dichtonic research)

في الحالة الأسوأ سيتم تنفيذ الحلقة RS من البرنامج لعدد يساوي N − 2 مرّة ، مما يعني إن مدة التنفيذ هي بحدود العدد N . من هنا نرى إنه من المفيد بل من المفروض تخفيض مدة تنفيذ هذه الحلقة .

9.1.2.1 ـ صيغة البحث الفرقاني

لنفترض إن V عبارة عن جدول بعناصر مرتبة حسب الترتيب التصاعدي وX عبارة عن عدد ، فلنبحث عما إذا كان العدد X هو عنصر من الجدول V .

صيغة البحث الفرقاني :

نقارن X مع العنصر المركزي من الجدول (يُدعى هذا العنصر محور) ، فمن الممكن أن نعرفه إذا كان العدد موجود في القسم الأيسر أو في القسم الأيمن من الجدول . نبدأ أولاً بالبحث في نصف الجدول الأيسر أو الأيمن . نكرِّر هذه العملية حتى نحصل على جدول ثانوي مؤلف من عنصر واحد أو حتى نلتقي بالعنصر الذي نبحث عنه .

مثلاً :

X = 8

V = 1, 3, 5, [6] 7, 8, 10, 12, 14

القسم الأيسر: 1, 3, 5 — محور: 6 — القسم الأيمن: 7, 8, 10, 12, 14

البحث يبدأ في القسم الأيمن لأن 8 العدد X = 8 هو أكبر من المحور 6 .

7	8	[10]	12	4
قسم أيسر		محور	قسم أيمن	

البحث يبدأ في القسم الأيسر لا من 10 > 8 .

[7]	8
محور	

البحث يبدأ في القسم الأيمن لأن 7 < 8 ويقف لأن الجدول الجديد يتألف من عنصر واحد .

يمكن للبحث أن يقف قبل أن نحصل على أي جدول جديد لأن العنصر الـذي نبحث عنه هو عبارة عن محورَ . في المثل السابق ، لو كان العدد X يعادل 10 ، لكان البحث قد توقف في المرحلة الثانية .

بإمكاننا تحسين البرنامج بملاحظة إن المتحولة i لم تتعدَّل إذا كانت المتحولة a هي تحديداً أصغر من L(y) في جسم الحلقة . فمن غير المفيد إذاً إعادة تقييم للاختبار $a \geqslant L_{(i+1)}$ ، مما يعطي :

```
RD If a ≥ L(n) then i : = n
   else if a < L(1) then i : = 0
        else
Begin i : = 1; s: = n;
        While a > L(i + 1) DO
        Begin J : = (i + s) div 2;
        While a < L(J) DO
        Begin s : = j; j: = (i + s) div 2 END;
        i : = j
        end;
        end;
present: = if i = 0 then false
      else if a = L(i) then true
        else false
```

8.1.3 ـ البحث الفرقاني ـ برنامج بلغة باسكال

لنفترض وجود جدولاً من 100 عنصر مرتبة بترتيب تصاعدي . نرغب بمعرفة فيما إذا كان أحد العناصر يعادل 38.5 ؛ (الإنتباه : لا يمكن أن يتم الاختبار بالتعادل الدقيق مع الأعداد الحقيقية ، بسبب الدقة المحدودة للحاسبات) . عمليات الاختبار تتم على الشكل التالي :

ABS (x − y) < Z أو (x > y − T) AND (x < y + T)

سنقوم قليلاً قليلاً بتقسيم الفسحة المريبة على 2 .

```
J : = (i + s) div 2;
if a < L(j) then s : = j else i : = j;
```

أي إذا كان A < L(j) سنبحث عن A في الفسحة الممتدة بين i وj ، والمتحولة S ستأخذ قيمة J ودور N . أما إذا كان A > L(j) فالبحث سيتم بين J وN ، وعند ذلك سيأخذ مؤشر الانطلاق بالبحث i القيمة j والبحث سيتم بين J وN .

التحليل :

S2 : طالما إن العنصر غير موجود والجدول الثانوي يحتوي على أكثر من عنصر ، كرِّر البحث .

S21 : إحسب مؤشر المحور
إذا كان العدد الذي نبحث عنه أصغر من المحور .

إذن S211 : البحث سيتم من اليسار .
وإلا S212 : إذا كان العدد الذي نبحث عنه أكبر من المحور
إذن S2121 : البحث سيتم من اليسار .
وإلا S2122 : العدد موجود .

لنعود الآن إلى المثل السابق . ولنأخذ الحالة الأسوأ .
لنفترض إن قيمة المؤشر (الذي عند بلوغه سنجد قيمة A) هي :

$$\text{ind}(A) = N - 1$$

عند البحث بداخل الفسحة [i.N] ، وبعد كل خطوة أو كل عملية بحث سنجعل i تأخذ القيمة [i + N] ، وهي قيمة وسطية في الفسحة [i, N] ، أي سنُقسِّم الفسحة

[i, N] إلى فسحتين ونبحث في كل فسحة منها على حدى ، فإذا لم يكن العدد A موجود في الفسحة الأولى يكون حكماً في الفسحة الثانية . هكذا ، وعند بلوغ إحدى القيم الوسطية j ، فقد نحصل على اللامعادلة A < L(j) . من هنا سنقوم بإدخال متحولة جديدة إلى البرنامج هي S ، وهذه المتحولة تأخذ في البداية القيمة N ، وسنستعمل المعادلة :

$$L(i) \leqslant A \leqslant L(s)$$

المحفوظة في الحالة التي يكون فيها A < L(j) ، وبواسطة تخصيص j إلى S ، سيصبح جسم الحلقة إذن :

مثلاً، في الحالة الأولى، سنفحص العنصر TAB[50] . وإذا كان TAB[50] =x، فمعنى هذا إننا إنتهينا ، وإلا فإذا كان TAB [50] > x ، فمعنى هذا إنه لم يبقَ لدينا سوى البحث في الفسحة من 1 إلى 50 . وإذا كان TAB [50] < x ، لا يبقى لنا سوى البحث في المجال من 50 إلى 100 .

البرنامج :

```
CONST N = 100;
          X = 38.5;
VAR INT, SUP, IT : INTEGER;
     TROUVE : BOOLEAN;
     TAB : ARRAY [ 1.. N ] OF REAL;

INF: 1;
SUP: N;
TROUVE : FALSE;
REPEAT
IT : = (INF + SUP) DIV2;                                  ( الحدود الجديدة )
IF (TAB [ IT ] > x - 0.0001) AND ( TAB [ IT ] < x + 0.0001)
  THEN TROUVE : = TRUE
  ELSE
     IF [ IT ] < x THEN INF : = IT + 1
                   ELSE SUP : = IT - 1
.UNTIL TROUVE OR (INF > SUP);
```

8.2 ـ الفرز TRI

عملية الفرز هي عملية تقوم على تصنيف وإيجاد إحدى الفقرات أو المعطيات أو إحدى الكلمات من داخل مجموعة معينة .

مثلاً : أجد العنصر الأكبر من داخل جدول من الأعداد الصحيحة الإيجابية .

8.2.1 ـ الفرز بطريقة شل (Shell metode)

يتعلّق ذلك بطريقة الفرز « بالفقاعات » حيث ، ولترتيب أحد الجداول ، يتم تبديل كل عنصرين متقاربين، أحدهما بالآخر ، إذا لم يكونا في الترتيب الصحيح . سيتم فحص جميع الأزواج من العناصر ، وإذا جرى أي تبديل بين العناصر في إحدى عمليات المقارنة ، يتم إجراء عملية مقارنة جديدة .

هذه الطريقة هي عديمة الفعالية بالنسبة للجداول الكبيرة . سنقوم بإدخال طريقة جديدة للفرز هي طريقة شل «Shell Sort» أو طريقة Shell .

بدأ شل عمله من الملاحظة التالية : ما هو سيىء في طريقة الفرز « بالفقاعات » هو في كون البعد (ecart) بين العناصر المتبادلة هو دائماً صغير ويساوي دائماً 1 . لنفترض ، الحالة الأسوأ ، بأن العنصر الأكبر هو في رأس الجدول منذ البداية . ويجب أن يصبح في نهاية الجدول بعد الفرز . وبالتالي لا يمكن أن نجد هذا العنصر أو نفرز هذا الجدول وننظمه إلا بعد $n - 1$ عملية مقارنة ، و$n - 1$ عملية تبديل . بينما لو كنا قد أنشأنا أبعاداً تعادل $n/2$ ،لكنّا نحتاج إلى عمليتي مقارنة وعمليتي تبديل .

فطريقة شل إذن ، تقوم على أن نأخذ ، في البداية ، أبعاداً كبيرة بين العناصر المُتبادلة ، وبعد ذلك نقوم على تخفيض هذه الأبعاد . البعد الأول الذي نأخذه سيكون $n/2$ (n = حجم الجدول) ، وبعد ذلك سنقوم بتقسيم هذا البعد على 2 في كل مرة حتى يصبح البعد معادلاً لواحد .

لكل قيمة للبعد وبطريقة ما سنقوم بإجراء عملية الفرز « بالفقاعات » حيث المقارنة تتم بين [TAB [I و[TAB [I + ECART ، وحيث هذه العملية ستكون متداخلة ضمن حلقة على ECART .

هكذا فبرنامج الفرز حسب الترتيب التصاعدي باستعمال طريقة شل سيبدو على الشكل التالي :

```
PROGRAM SHELL;
    CONST NB = 50;                                  { عدد العناصر }
    VAR ECH;
      ECART,
      I : INTEGER;
      TAB : ARRAY [ 1.. NB ] OF INTEGER;

PROCEDURE ECHAN GE;
    VAR X : INTEGER;
    BEGIN
      X : = TAB [ I ] ;
      TAB [ I ] : = TAB [ I + ECART ] ;
      TAB [ I + ECART ] : = X

    END;                                            { نهاية التبادل }

BEGIN
WRITELN ('TABLE NON CLASSE');
    FOR I : = 1 TO NB DO
      BEGIN
        READ (TAB [ I ] );
        WRITE (TAB [ I ] : 5)
    END;

WRITELN ; WRITELN;
{ start of tri }
ECART : = NB;
REPEAT                                  { الحلقة على مختلف الأبعاد ECART }

   ECART : = ECART DIV 2;
   REPEAT
```

```
            ECH = 0;
            FOR I : = 1 TO NB – ECART DO
BEGIN
  IF [ I ] > TAB [ I + ECART ]
    THEN BEGIN
            ECH: = 1;
            ECHANGE
            END
END
    UNTIL ECH = 0
UNTIL ECART = 1;
WRITELN ('TAB LEAV CLASSE');
FOR I : = 1 TO NB DO
    WRITE (TAB [ I ] (: 5);
```

E8.3 ـ تطبيقات على طريقة البحث الفرقاني

حسابة الجذر التربيعي لعدد إيجابي أو صفر .

الحلّ :

حسب التعريف ، العدد الصحيح X هو الجذر التربيعي للعدد A إذا كانت المعادلة $x^2 \leq A < (x+1)^2$ صحيحة .

من هنا نستخرج مباشرة البرنامج R1 ، الذي يستعمل الحلقة $0 \leq x^2 \leq A$ ، التي يدوم تنفيذها حوالي [$\sqrt{A}$] .

```
R1 : X : = 0;
    While (x + 1) ↑ 2 ≤ a DO x : = x + 1
```

إذا كان هناك قيمة d تستطيع أن تأخذ قيمة أكبر مباشرة من $\sqrt{A}$ ، فبإمكاننا أن نتبع الطريقة المتبعة في البحث الفرقاني :

سنقوم باختيار القيمة له ، بشكل $d > [\sqrt{A}] \geq x$) نختبر d بالنسبة لـ $x = 0$

و$A \sqrt{}$ < d) ، فالاختزال اللوغاريتمي يقوم على تخصيص x أو d بالقيمة الوسطية] + x
. [d

الخروج من الحلقة يتم عندما يكـون معنا : d − x = 1 . من هنـا نحصل عـلى البرنامج التالي :

```
R2 : x : = 0;
     While d − x > 1 DO
     BEGIN
J  : (x + d) div 2;
if j ↑ 2 > a then d : = j
            else x : = j
end
```

8.4 ـ الفرز بالتجزئة (Sort by seymentation)

8.4.1 ـ خوارزم الفرز

عملية فرز أحد الجداول تقوم على تـرتيب عناصر الجـدول بنظام تـرتيب محدَّد . لنفترض إن الجدول المطلوب فرزه A(1::N) يحتوي على أعداد صحيحة ويجب إعادة ترتيبها حسب النظام التصاعدي للأعداد .

لنأخذ المؤشرات j، i بحيث $i \geqslant 1$ و$j \leqslant N$. سنرمز إلى الجدول الثانوي بالمؤشرات i وj على الشكل التالي A (i:: j) .

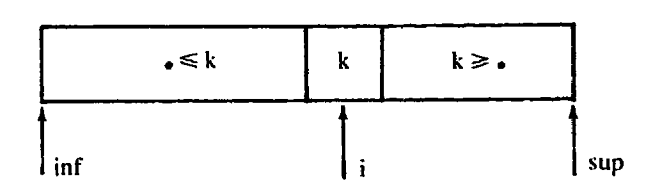
والآن لنفترض المسألة pinf, sup : « إفرز الجدول A(inf:sup) » . يرتكز الفرز بالتجزئة على التقسيم المتتالي التالي :

أ ـ إذ كان $inf \geqslant sup$ ، فالجدول A(inf::sup) سيكون فارغاً أو يحتوي على عنصر واحد عندها فهو مفروز.

ب ـ إذا كان $inf < sup$.

1 ـ نبدأ بتبديل عناصر الجدول A(inf::sup) حتى نصل إلى مؤشر i بحيث : لكل مؤشر جديد l ، $i > l \leqslant inf$ ، فنحصل على $A(l) \geqslant A(i)$ ، ولكل m ، $i < m \leqslant$ sup ، نحصل على $A(i) \leqslant A(m)$. سنشير إلى القيمة A(i) بواسطة العدد k .

هذه الصفة يُمكن أن تتمثَّل بواسطة :



هذه المرحلة تدعى : تجزئة الجدول A(inf::sup)

2 ـ في الجدول المفرو ، ستكون قيمة العنصر A(i) معادلة لـ k . وبعد التجزئة ، يكفي بأن نقوم بحلّ المسألتين P inf, i − 1 وP i+1, sup ليصبح الجدول مفروزاً .

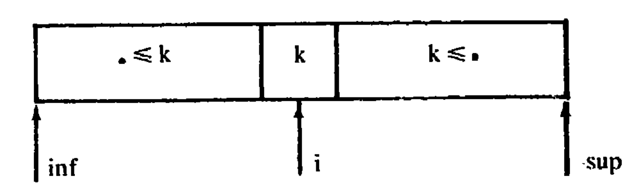
من هذه التجزئة نحصل على الاجراء المتكرِّر التالي :

```
Procedure IRI ( Integer Value inf, sup);
   If inf < sup then
      Segmentation (inf, sup, i);
      tri (inf, i − 1); tri (i + 1, sup)
   end
```

8.4.1.1 ـ كتابة إجراء التجزئة

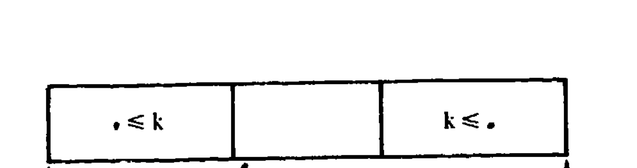
يُطبَّق إجراء التجزئة على الجدول A (inf::sup) (inf < sup) . الشرط المسبق للتجزئة هو :

يوجد مؤشر i بحيث :



يمكن التحقُّق من هذا الشرط المسبق بالنسبة لأية قيمة لـ k تظهر في الجدول A(inf::sup) . بإمكاننا إذاً ، إختيار وبشكل عشوائي القيمة K من ضمن عناصر الجدول A(inf::sup) وذلك قبل التجزئة . تدعى هذه القيمة « مدار » (PIVOT) التجزئة . سنأخذ k = A(inf) .

ومن الممكن إستعمال الحلقة التالية :



أي :

(inf ≤ l ≤ sup) and (inf ≤ j < l →
(A, J) ≤ k)) and (m < j ≤ sup → (A(J) ≥ k))

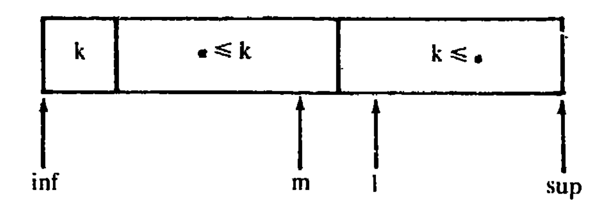
لاختزال عدد عمليات التبادل ، فلنكتب حلقة من البرنامج :

```
(1) While l ≤ m DO
    if A(l) ≤ k then l : = k then l : = l + 1
    else Begin
        While (A (m) > k) and (l < m) DO
          m : = m − 1;
          A(l) : = : A(m);
          m: = m − 1
      end
```

يتم وقف الحلقة 2 بواسطة الاختبار A(m) < K لأننا إخترنا k = A(inf) ، والاختبار l < m يجب أن يتم تقييمه بعد هذه الحلقة ؛ لأن التجزئة تكون قد إنتهت إذا حصلنا على l > m .

من جهة أخرى ، وبعد تبادل A(l) وA(m) ، نحصل على A(l) ≤ k ، إذن باستطاعتنا أن نزيد قيمة l واحد (l = l + 1) .

الشرط المسبق للحلقة هو :



لنحصل على الشرط المسبق للتجزئة ، يكفي أن نستبدل القيم A(inf) وA(m) .

من هنا نحصل على الاجراء التالي :

```
procedure segmentation (integer value inf, sup: integer variable m);
  Begin integer l, k;
    l: = inf + 1; m: = sup; k : = A (inf);
    while l ≤ m DO
      if A(l) ≤ k then l : = l + 1

else A(l) ≤ k then l: = l + 1
else Begin
   while A(m) ≥ k DO m = m − 1;
      if l < m then
Begin
A(l) : = : A (m);
m: = m − 1;
l : = l + 1
end.
end;
(A(m) : = : A(inf)
end.
```

الفصل العاشر

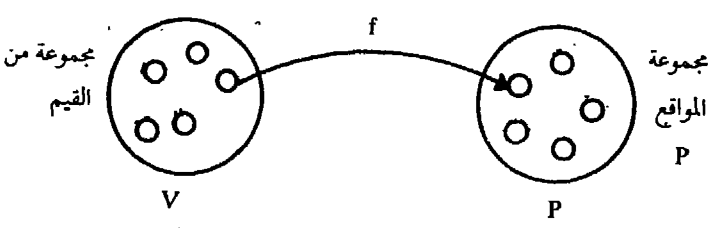
السجلات (files)

10.1 ـ تعريف :

ـ لنفترض المجموعة V ، هذه المجموعة هي عبارة عن مجموعة من العناصر أو « القيم » تؤلف السجل المتتالي .

ـ المجموعة المحدَّدة والمنظمة حسب ترتيب معين P ، هذه المجموعة تُشكِّل مجموعة مواقع السجل المتتالي .

السجل المتتالي هو عبارة عن تطبيق f للقيم V في P .



وبشكل عام ، فإن السجل هو عبارة عن مجموعة مُنظَّمة من المعلومات . تختلف المعلومات المخزَّنة في السجل بنوعيتها وبتركيبها ، وتكون عادة مُقسَّمة إلى تسجيلات متشابهة بالتركيب ومختلفة بالمضمون (أي بالمعطيات المخزنة فيها) .

	record	تسجيلة	ـــ ـــ ـــ ـــ

سجلّ (file)

مثلاً :

```
type personne = struct
    name : chain 20;
       adress : chain 40;
```

```
        married : boolean;
        sons: integer;
END;
```

سنقوم هنا بتعريف نوع (type) مُركَّب يُدعى شخص (personne) ، يتألف من « إسم » (name) (سلسلة من 20 سمة (char)) ، من « عنوان » (adress) (سلسلة من 40 سمة (character) على الأكثر) ، من متحولة منطقية (boolean) تدعى « متأهل » (married) ، ومن عدد صحيح يدل على عدد « الأولاد » (sons) .

نقول إن الشخص (personne) يحتوي على أربعة حقول (fields) ، هي : «name» ، «adress» ، «married» ، «sons» . ولاختيار وبلوغ أحد هذه الحقول نستعمل التعبير التالي :

«name of variable» . «name of field»

« إسم الحقل » . « إسم المتحولة »

مثلاً :

إذا كانت المتحولة pers هي متحولة من نوع «personne» ، سنكتب : pers.name ، pers. adress ،

pers. married . pers. sons ،

لبلوغ مختلف حقول أو متحولات هذه المتحولة المركبة pers .

10.2 ـ تعابير خاصة في إستعمال السجلات

أ ـ نستعمل عند كتابة الخوارزميات هذه الرموز للإشارة إلى السجلات :

- f^+ : تعني السجل الثانوي المقروء .
- f^- : تعني السجل الثانوي الباقي للقراءة .
- العنصر الجاري هو العنصر الأول من السجل f^+ .

ب ـ إشارة نهاية السجل

نهاية السجل eof (end of file) .

eof تأخذ القيمة true إذ بلغنا نهاية السجل .

ج ـ بلوغ العنصر الأول من السجل .

reread (file name)

هذه العملية تتيح لنا بلوغ العنصر الأول من السجل إذا كان موجوداً . وهي تُركِّز الشريط بشكل يكون فيه رأس القراءة / الكتابة في مواجهة العنصر الأول من السجل كي تسمح بقراءته . في حالة السجل الغير فارغ ، تكون قيمة العنصر الأول من السجل مُرتبة في حيِّز العمل (work jone) الذي يُدعى حيز الداريء (Buffer zone) .

د ـ أمر بلوغ العنصر التالي :

gef (file name)

هـ ـ أمر إنشاء السجل الفارغ .

rewrite (file name)

و ـ أمر إضافة العنصر

put (file name)

لا يمكن أن نستعمل هذا العنصر إلاّ إذا كان رأس القراءة مُركّزاً في نهاية السجل .

أي : eof = true .

تأويل هذا الأمر يبدو على الشكل التالي :

ـ نسخ ، في نهاية السجل ، للقيمة الموجودة في حيِّز الداريء المرتبطة بالسجل .

ـ تقدُّم الشريط لموقع ، كي يتم تركيز رأس الكتابة على إشارة نهاية السجل .

ـ بعد تنفيذ هذا الأمر ، تأخذ المتحولة eof القيمة true .

ـ قيمة مضمون حيز الداريء f هي دائماً غير مُحدَّدة .

مسألة :

البحث عن القيمة القصوى الموجودة في أحد السجلات .

نبحث عن القيمة m ، $m \notin f$ ، التي ومهما يكن $\forall y \in f$ ، نحصل على $m>y$.

هكذا قيمة تدعى القيمة القصوى أو maximal .

```
procedure maximum (df : file of t; r max: t);
    specifications { f = <> } → { max 6 f, max ≥ f) }
    Begin
```

```
reread (f); { eof (f) }
max : = f';
get (f); { max 6 f, max ≥ f }

while NOT eof (f) DO
Begin
if max : = f';
get (f)
{ max 6 f  , max ≥ f }
end
{ (eof (f), max 6  , max ≥ f ) → (max 6 f, max ≥ f)}
end;
```

من الممكن أيضاً كتابة هذا الاجراء على شكل دالة (function)

```
function maxi (df : file oft ) : t;
     specifications { f = < > } → { max 6 f, maxi ≥ f }
var max: t;
Begin
reread (f); max : = f; get (f);
while NOT eof (f) DO
Begin
if max < f' then
     max : = f'
     get (f)
end;

maxi : = max
end;
```

النوع t هو منطقي ، صحيح ، من نوع رمزي (char) ، أو من نوع سلسلة من السمات (chain of char) .

10.3 ـ التصريح عن السجلات بلغة باسكال

بلغة باسكال ، التصريح عن السجل المتتالي يتم حسب التعبير التالي :

```
VAR ARTICLE : TYPE ARTICLE;
    FICHIER1 : FILE OF TYPE ARTICLE;
```

وهـذا مـا يُنشـىء سجـل متتـالي FICHIER 1 ، عبـارة عـن متتالية مُنظَّـمة من المواضيع أو الأشياء من نوع TYPARTICLE . ولأن المتتالية هي محدَّدة ، فإن الموضوع الأول والأخير هما محددان . يمكن للسجلات أن تكون مشكَّـلة من مواضيع مختلفة النوع ، وقد تكون عبارة عن سجلات من نوع سجلات أو ما يسمى بـ : file of file .

ولكن ، في أكثر الأحيان ، تكون السجلات عبارة عن سجلات من السمات FILE OF CHAR ، أو سجـلات من نـوع معـين FILE OF TYPEARTICLE حيـث TYPEARTICLE عبارة عن تسجيلة (record) معينة .

يـدعى العنصر المبلوغ من السجـل FILE ↑ (أو FILE @) ، يلعب العنصر FILE ↑ دور معـرِّف الداريء في الـذاكرة القـادرة على إحتـواء التسجيلة (record) . العمليات التي تتم على السجلات تقوم على التبادل الفيزيائي بين الداريء (Buffer) . والجهاز المحيطي (peripherique) الناقل للسجل .

لتعبئة الداريء ، يمكن أن نكتب .

```
FILE ↑ : = ARTICLE;
```

حيث ARTICLE عبارة عن عنصر أو تسجيلة من السجل .

ولاستعمال الداريء (بعد القراءة) ، يمكن أن نكتب :

```
ARTICLE : = FILE ↑
```

10.4 ـ كتابة السجل .

إنشاء السجل يتم بواسطة :

```
REWRITE (FILE1);
```

هذا الإنشاء يُركِّـز بداية السجل ، وبعد ذلك نكتب :

```
FILE 1 ↑ : = ARTICLE;
    PUT (FILE 1);
```

هذه التعليمات تكتب التسجيلات أو المعلومات ARTICLE بشكل متتالٍ في السجل .

يمكن جمع العمليتين الأخيرتين باستعمال الاجراء النموذجي :

WRITE (FILE 1, ARTICLE);

مما يؤدي إلى تفادي إدارة الداريء FILE ↑ . يجب أن يكون العنصر الثاني ARTICLE (وهو يعني ما يجب كتابته) حكماً أو إلزامياً من النوع الأساسي للسجل .

10.5 ـ قراءة السجل

يجب أولاً نداء الإجراء النموذجي التالي :

Reset (file1)

يؤدي هذا الإجراء إلى العودة إلى بداية السجل وإرسال العنصر الأول الى الداريء . بعد ذلك ، تتم عمليات القراءة بواسطة سلسلة من التعليمات :

ARTICLE : = FILE 1 ↑ ;

GET (FILE 1);

تؤدي GET إلى إرسال العنصر الموجود أمام رأس القراءة إلى الداريء (Buffer) . وسيتم إستعمال الداريء في التعليمات اللاحقة .

ARTICLE : = FILE1;

GET (FILE 1);

أي وبكلمة أخرى ، عند القراءة ، يجب أن نكون دائماً مُتقدِّمين بعملية GET واحدة ، وهذه هي GET الموجودة في RESET . هذا التقدم إلى الأمام يسمح باكتشاف نهاية السجل .

في هذه اللحظة ، يكون FILE 1 ↑ غير محدَّد ، والدالة المنطقية (EOF (FILE 1 ↑ تصبح حقيقية TRUE .

نفس الأمر ، وكما بالنسبة للكتابة ، سيتم إستبدال زوج التعليمات ;...=:ARTICLE و ;...GET ، بإدخال الاجراء النموذجي التالي :

READ (FILE 1, ARTICLE);

فلنشر إلى إن READ وWRITE يُمكن أن تُطبَّق على عدة معطيات متتالية . مثلاً :

READ (FILE, ARTICLE1, ARTICLE2, ARTICLE3);

10.6 ـ جداول الجداول ، أو الجداول المتعددة الأبعاد

إضافة إلى السجلات ، فإن لغة باسكال تعرِّف عن الجداول المتعددة الأبعاد (multi dimension array) والنوع record .

النوع الأساسي للجدول يمكن أن يكون بنفسه عبارة عن جدول . من هنا نحصل على :

```
type VECTOR = ARRAY [ 1..N ] OF REAL;
     MATRICE = ARRAY [ 1..N ] OF VECTOR;
     Var V : VECTOR;
        M: MATRICE;
```

يُمكن أن نتعرَّف على عنصر المصفوفة MATRICE بواسطة :

M [I] [J] : = 3;

ـ M : عبارة عن مُتَّجه (Vector) . تتألف عناصر هذا المتجه M(J) من المصفوفة MATRICE المصرَّح عنها بواسطة Var ، حيث كل عنصر منها بدوره عبارة عن جدول من الجداول VECTOR . بإمكاننا إذاً أن نكتب :

M [J] := V;

وبشكل عام ، فإن التصريح عن جدول الجداول يتم على الشكل التالي :

```
TYPE identificator = ARRAY [ 1..N ] OF type;
     identificator 1 = ARRAY [ 1..N ] OF identificator
          Var V : identificator
               M : identificator 1
```

ـ النوع TYPE : عبارة عن جدول من N عنصر من النوع type .

ـ النوع identificator 1 : عبارة عن جدول من N عنصر من النوع identificator ، أي إن كل عنصر من هذا الجدول هو عبارة عن جدول من N عنصر من النوع Type .

ـ المتحولة V المصرَّح عنها بعد الكلمة Var ، تُمثِّل متحولة من النوع identificator .

ـ المتحولة M : فهي من النوع identificator I ، وبالتالي فهي عبارة عن جدول بعناصر تُشكِّل بحد ذاتها جداول من النوع type .

من الممكن أيضاً التصريح عن الجداول المتعددة الأبعاد ، بواسطة :

```
TYPE X = ARRAY [ S ] OF ARRAY [ S ] OF ARRAY [ S ] ;
```

ومن الممكن أيضاً التصريح عن المصفوفة على الشكل التالي :

```
TYPE MATRICE = ARRAY [ 1..N, 1..N ] OF REAL;
```

مسألة :

لنفترض المصفوفات A وB ، فلنحسب المصفوفة C = A × B .

بما إن :

A = [1..N; 1..N]

B = [1..N, 1..N]

فنتيجة ضربهما ستكون عبارة عن مصفوفة C بأبعاد [N .. 1, N..1] C . عناصر هذه المصفوفة تعادل :

$$C_{ij} = \sum_{k=1}^{n} A_{ik} B_{kj}$$

البرنامج سيبدو على الشكل التالي :

```
PROGRAM PRODM A;
    CONST N = 10;
    TYPE MATRICE = ARRAY [ 1..N, 1..N ] OF REAL;
        VAR A, B, C : MATRICE;
        I, J, K : 1.. N;
BEGIN
  FOR I : = 1 TO N DO
    FOR J : = 1 TO N DO
      BEGIN
        C [ I, J ] : = 0.0;
          FOR K : = 1 TO N DO
          C [ I, J ] : = C [ I, J ] + A [ I, K ] × B [ K, J ]
      END
  END.
```

10.7 ـ النوع تسجيلة (record type)

الجدول هو الوسيلة لتجميع عدد n من العناصر من نفس النوع تحت إسم واحد . مثلاً ، أرقام العشرة سنوات الأخيرة ، أو أسماء جميع الزبائن . ولكن من الضروري في بعض الأحيان تجميع العناصر من أنواع مختلفة في موقع واحد وتحت نفس الإسم الواحد .

هذا هو بالتحديد ما يحصل في التسجيلات التي تؤلف السجلات . مثلاً : يجب أن نجمع لكل زبون من سجل الزبائن المعلومات التالية :

المعلومات	إسم المتحولة	معنى المتحولة
ـ رقم الزبون	NUMERO	numero
ـ إسم الزبون	NOM	nom
ـ عنوان الزبون	ADRESSE	adresse
ـ كود البريد	CPVILLE	code postal
ـ التمثيل الذي يشغله	NUMREP	representation
ـ إذا كان له حسم	REMISE	remise
ـ رقم عمله في السنوات السابقة	CAPREC	chiffre d'affaires
ـ رقم عمله في السنة الحالية	CA COURS	nouvelle chiffre d'affaire

لصياغة هذه المعلومات ، تتمتع لغة باسكال بالنوع RECORD . هكذا ، فالتصريح عن المعلومات أعلاه تتم على الشكل التالي :

```
TYPE CLIENT = RECORD
            NUMERO   INTEGER;
    NOM          : PACKED ARRAY [ 1..10 ] OF CHAR;
    ADRESSE      : PACKED ARRAY [ 1..30 ] OF CHAR;
    CPVILLE      : PACKED ARRAY [ 1.. 20 ] OF CHAR;
    NUM REP      : INTEGER;
    REMISE       : BOOLEAN;
    CAPREC       : REAL;
    CA COURS     : REAL;
END.
```

من الممكن إذاً أن نصرِّح عن المتحولة CLI ، وكأنها من النوع CLIENT ، أي تحتوي على نفس المعلومات الواردة في النوع RECORD .

VAR CLI.CLIENT;

وبالإمكان أن نبلغ أي عنصر من CLI على الشكل التالي :

CLI . NOM : = « DUPONT␣␣␣ »;

وأن يتم إجراء أية عملية مقارنة أو غير ذلك على الشكل التالي .

مثلاً :

IF CLI.REMISE THEN..

وبالإمكان التصريح عن النوع جدول من التسجيلات RECORD على الشكل التالي : لنفترض النوع RECORD الخاص بالزبائن CLIENT ، وإذا كنا نرغب بالتصريح عن جدول من العناصر CLIENT ، فذلك سيتم على الشكل التالي :

TYPE ENTREPRISE = VAR [1..50] OF CLIENT;
VAR E1 : ENTREPRISE;

حسب هذا التصريح ، سيكون رقم العمل للزبون الخامس من الشركة E1 (ENTREPRISE) هو :

E1 [5] . . CA COURS

والحرف الأول من إسم هذا الزبون الخامس . سيكون :

E1 [5] . NOM [1]

10.7.1 ـ التعليمة WITH

بدلاً من كتابة المعلومات التالية للإشارة الى مختلف أجزاء التعليمة :

CLI . NUMERO : = ...
CLI . NOM : = ...
CLI . ADRESSE : = ...
CLI ....

فالتعليمة WITH تسمح بتفادي تكرار CLI (أو أي معرِّف آخر) ، وللاشارة إلى المعلومات السابقة نستطيع أن نكتب :

```
WITH CLI DO
    BEGIN
        NUMERO : = ...
        NOM : = ...
        .
        .
        .
        END;
```

وفي الحالة التي يوجد فيها عدة مستويات من RECORD ، أي عدَّة معرِّفات .

مثلاً :

```
ENTREPRISE. USING. SERVICE . CLI . NOM : = ...
```

فباستطاعتنا أن نكتب ما يلي :

```
WITH ENTREPRISE, USINE , SERVICE , CLI DO
    BEGIN
    NOM : =
        END;
```

10.8 ـ سلاسل السمات في لغة باسكال

تتألف سلاسل السمات من مجموعات من الرموز الأبجعددية (alphanumeric) المنظمة حسب ترتيب معين ، وتؤلف كلمات معلوماتية معينة . ومن غير الممكن أن تُستعمل المتحولة الرمزية المؤلفة من سلسلة من السمات كمتغيِّر في الأمر Read ، ولكن لقراءة هكذا متحولة رمزية نستعمل عادة البرنامج التالي :

```
for i : = 1 TON do
    read (V [ i ] )
```

حيث V هي عبارة عن متحولة من السمات الرمزية ، مصرَّح عنها على الشكل التالي :

```
Var V : packed array [ 1..N ] of char;
```

الإجراء العام لقراءة سلسلة من السمات ، يُكتب على الشكل التالي :

```
procedure read chain (Var x: packed array  [ 1.. max : integer ]  of char)
 Label 1;
   Var car : integer;
   begin
     for car : 1 TO max do
       if eof then begin
       writeln ('eof while the chain is readen')
       go to label 1
     end
     else
       read (X [ car ] );
Label 1 : end;
```

الفصل الحادي عشر

تطبيقات (Applications)

11.1 ـ مسألة :

إحسب عدد المرّات التي نلتقي فيها الكلمة LEBANON في جملة واحدة .

الحلّ :

الجملة هي عبارة عن سلسلة من الكلمات المنفصلة عن بعضها بواسطة واحدة أو عدة فراغات . تنتهي الجملة بالسمة و.و (نقطة) .

الكلمة هي عبارة عن مجموعة من السمات الأبجدية المختلفة ، والمحصورة من جهة اليسار بواسطة فراغ واليمين بواسطة فراغ آخر (Space) أو نقطة .

تُكتب الجملة على الناقل المعلوماتي أو جهاز الإدخال disk ، tape) بشكل سلسلة من السمات ، حيث تُسَجَّل عناصر هذه السلسلة على التوالي بداخل متحولة رمزية تدعى car cour .

وعلى العكس ، فإن عرض المسألة يُوحي لنا بتقسيم آخر للجملة : الجملة هي عبارة عن سلسلة من السمات المفصولة عن بعضها بواسطة فراغات ، والكلمات عبارة عن سلاسل من السمات المختلفة عن الفراغ والنقاط . من هنا ، نحصل على المسودة الأولى للخوارزم .

```
Begin
  car blanc is '' , car marq '.'
  int lg is 7; { طول الكلمة التي نبحث عنها }
  [ 1: lg ] char word is ('L', 'B', 'A', 'N', 'O', 'N');
  Var car car cour init read car;
```

{ المتحولة car cour تسمح لنا بالبحث وعبور سلسلة السمات التي تؤلف الجملة }

{ المتحولة carcour = فراغ أو نقطة أو تعادل السمة الأولى من الكلمة الأولى }

TO end of sentence **DO**

{ المتحولة carcour = فراغ أو نقطة أو السمة الأولى من الكلمة الأولى }

While carcour = marq sort by end of sentence;

{ carcour = السمة الأولى من الكلمة }

treat word

continue

{ carcour = فراغ أو نقطة }

results

END

شروحات :

ـ المتحولة carcour هي المتحولة الرمزية التي نستقبل السمات بعد قراءتها من الناقل .

ـ المتحولة word تمثّل الكلمة التي نبحث عنها ، وتأخذ في البداية القيمة LEBANON .

ـ نُعرِّف عن المتحولة فراغ (' ') بالاسم blank ، وعن المتحولة marq التي تعني النقطة ..

ـ يجري التصريح عن المتحولة الرمزية carcour على إنها رمزية char ، وتأخذ السمة المقروءة read car في كل مرّة .

ـ end of sentence وتعني نهاية الجملة .

ـ treat word وتعني إجراء لمعالجة الكلمة التي نبحث عنها بعد تجميعها وإيجادها .

2 ـ إكتب الخوارزم الذي يسمح بالقفز فوق الفراغات الفاصلة للكلمات

Proc Saut blancs is

Begin

{ carcour = فراغ أو نقطة أو السمة الأولى من الكلمة الأولى }

TO end of blancs **DO**

while carcour = blanc **sort by** end of blancs;

carcour : = read car

continue

{ carcour = نقطة أو السمة الأولى من الكلمة الأولى }

end;

3 ـ إحسب عدد الكلمات الموجودة في أحد القواميس والتي تنتهي بالأحرف TION أي بقراءة الكلمات واحدة بعد الأخرى إحسب تلك المنتهية بـ TION .

التحليل :

S 2 : المعالجة

قراءة المجموعة الأولى من الكلمات من ثلاث كلمات .

طالما يوجد أيضاً مجموعات من ثلاث كلمات كرِّر .

S 21 : معالجة المجموعة من ثلاث كلمات .

طالما يوجد كلمة واحدة في المجموعة كرِّر .

S 211 : البحث بداخل السلسلة في الكلمة .

طالما لم تنتهي الكلمة كرِّر .

S 2111 : إذا وجدنا السلسلة .

إذن S 21111 : إحسب السلسلة (عدِّد السلسلة)

إعبر إلى نهاية الكلمة .

وإلا S 21112 : إعبر إلى الحرف التالي .

إعبر الى الكلمة التالية .

إعبر إلى المجموعة التالية .

الخوارزم :

Write «number of words finished by TION»

{ إكتب عدد الكلمات المنتهية بـ TION }

{ تحضير المعالجة }

Var chain word [26] : char

{ الكلمة الأطول في القاموس هي بطول 26 حرفاً }

Var counter : integer init 0;

Begin { counter = 0 }

```
IN = Word rang "AAAAAAAA"
                        { الكلمة الأخيرة من القاموس الفرنسي هي ZYTHUM }
While = word (1,6) < "ZYTHUM" Repeat
read # word                          { إقرأ كلمة جديدة بالأحرف الكبيرة }
end word : = 1
while (end word < = 26) and ( # word (end word, 1) < > "     ") Repeat

If # word (end word - 4, 4) = «TION»
  Then S 212 : Counter = Counter + 1
     ELSE
     end
end
$3 { writing of result }
     write { " number of words finished by TION = " }
     COUNTER
  end;
end.
```

ملاحظات :

أ ـ الكلمة الأطول في القاموس الفرنسي هي : AUTI CONSTITUTIONNELLEMENT وتتألف من 26 حرفاً .

ب ـ الكلمة الأخيرة من القاموس هي ZYTHUM (راجع القاموس petit robert) .

ج ـ تنفصل الكلمات عن بعضها بواسطة فراغات .

د ـ الكلمات التي تتألف من أطوال مختلفة ، يجب مراجعة طولها وأخذه بالحسبان . لذلك نعبر (نبحث) هذه الكلمات حتى بلوغ أول فراغ ، ومن خلال طول الكلمة سيكون من الممكن مقارنة الأربعة أحرف الأخيرة مع «TION» .

المتحولات المستعملة :

ـ Chain word : التصريح عن المتحولة الرمزية بالطول الأكبر وهو 26 سمة .

ـ end word : عبارة عن مؤشر يدل على السمات بداخل الكلمة .

ـ counter : عداد يحتوي على عدد الكلمات التي تنتهي بـ TION . وتزداد قيمته في كل مرّة نلتقي فيها كلمة تنتهي بـ TION .

ـ البحث يبدأ من نهاية الكلمات وإلى الوراء أربعة سمات :

if # word (end word − 4, 4) = «TION»

أي المؤشر end word ناقص 4 وبطول 4

مثلاً :

```
                    ↓    4
                    ├──────────→
A   A   A   A  |A   A   A   A
                |←───────────
                  end word − 4
```

عندما نلتقي بالكلمة TION يجري زيادة قيمة العداد Counter .

counter = counter + 1

ـ قراءة الكلمة تتم بواسطة التعليمة :

read # word

ـ التصريح عن الكلمة word على إنها سلسلة من السمات ، جرى في بداية البرنامج بواسطة التعليمة :

Var chain word (26)

26 يعادل طول الكلمة أو العدد الأقصى للسمات .

ـ { prossing } S ـ تعني البدء بالمعالجة .

11.3 ـ عمليات التكرار المتداخلة

مسألة : فوترة الزبائن

المطلوب كتابة برنامج يصنع الفواتير لجميع زبائن أحد المخازن التجارية .

الفاتورة هي عبارة عن نصّ يتألف من سلسلة السمات يدلّ على السلع ، أسماء الزبائن ، والمبالغ المستحقة .

نموذج الفاتورة هو عبارة عن نموذج متكرِّر ، تجري صياغته لكل زبون من الزبائن . وتحتوي على قسم رئيسي خاص بالشركة ، وقسم خاص بالزبون ، وقسم خاص بالسلع والمبالغ المستحقّة .

facture for client from 1 to nbcl.

nbcl : عدد صحيح يعادل عدد الزبائن .

وبما إن قيمة الضريبة TVA والتاريخ لا يتعلقان أبداً بالزبون ، لذلك يجب ذكرهما في النموذج الأساسي للفاتورة . من هنا نحصل على خوارزم بثلاث مستويات : المستوى الرئيسي (يشير إلى مركز الخدمة الخاص بإصدار الفواتير) . مستوى الزبون (أو الفاتورة الخاصة بالزبون) ، هذا القسم يتعلّق بالمعلومات الخاصة بالزبـون . مستوى السلعة المطلوبة ويحتوي على معلومات عن السلعة ، سعرها ، الكمية المطلوبة منها ، المبلغ الاجمالي لكل سلعة إلى ما هناك .

القسم الأساسي من الفاتورة

النموذج الرئيسي للفاتورة ، يحتوي على :

- فاتورة (نص) : يتألف النص من مجموعة فواتير الزبون .
- nbcl (عدد صحيح) يشير إلى عدد الزبائن .
- تاريخ (سلسلة سمات) إصدار الفاتورة .
- قيمة (عدد حقيقي) TVA لكل زبون .

خوارزم النموذج الرئيسي :

data = data ('DATE:')

nbcl = **data**

taux = **data** ('taux de la TVA:')

result = facture forclient from 1 to nbcl.

المعلومات الداخلة هي إذاً : التاريخ ، عدد الزبائن ، المبلغ ، ويجب إصدار الفواتير لكل زبون من 1 إلى nbcl .

صياغة الفاتورة لكل زبون من 1 إلى nbcl ، في تاريخ معين وقيمة ضريبة معينة تتم على الشكل التالي :

1 ـ تحليل المسألة . تحتوي الفاتورة على المعلومات التالية .

- السطر 1 (نص Text) : إستعلامات عن الزبون .
- السطر 2 (نص Text) : التاريخ .
- السطر 3 (نص text) : معلومات عن السلعة (en-tête) .
- لائحة بالسلع (نص) ، سطر لكل سلعة .

- recap (نص) : مجموعة من الأسطر ، يُكتب على كل سطر منها القيمة الاجمالية لكل سلعة من السلع .

المتحولات المستعملة هي :

- nom (سلسلة سمات) : اسم عائلة الزبون .
- pronom (سلسلة سمات) : إسم الزبـون .
- num (عدد صحيح) : رقم الزبون .
- rue (سلسلة سمات) : إسم الشارع .
- ville (سلسلة سمات) : إسم المدينة .
- NUP (عدد صحيح) : سلسلة ، الكمية المطلوبة .
- PV (عدد حقيقي) : سلسلة ، سعر الوحدة .
- PHT (عدد حقيقي) : سلسلة ، السعر بدون ضريبة .
- N (عدد صحيح) : عدد السلع .
- THT (عدد حقيقي) : المبلغ الاجمالي للفاتورة بدون ضريبة .
- TAXE (عدد حقيقي) : الضريبة أو قيمة TVA .
- TTC (عدد حقيقي) : قيمة الفاتورة أو المبلغ الاجمالي بما فيه جميع الضرائب .
- PRIX V : سعر الوحدة .
- PRIX HT : السعر بدون ضريبة .

الخوارزم :

facture = line 1 **To line** line 2 **to line** line 3

To line list of products to line recap

هكذا : فالفاتورة تحتوي على المعلومات التالية الرئيسية على الأسطر الثلاثة الأولى :

السطر الأول :	إسم الزبون	عنوان الزبون	رقم الزبون،	شارع ، المدينة
السطر الثاني :			التاريخ	
السطر الثالث:	رقم السلعة	الكمية	سعر الوحدة	السعر الاجمالي

1- Nom, prenom, numr, rue, ville = **data**

('nom, prenon, numero, rue, ville:')

2- N = data ('nombre d'articles commandés:')

{ عدد السلع المطلوبة : معطيات يجب إدخالها إلى البرنامج من الخارج }

3- ligne 1 = **write** nom, prenom, numer, ville

4- ligne 2 = write date

5- ligne 3 = write' NUMERO produit QUANTITE PRIX U, PRIX HT'

6- list of products = **write** NUP, Q, PV, PHT on line

for prod from 1 TON

PHT = Q × PV

7- TAXE = THT * TAUX

8« TTC = THT + TAXE

9- recap = write 'TOTAL HORS TAXE' , THT

WRITE »T.V.A', TAXE

write 'TOTAL. T.T.C', TTC

10- THT = **SUM of PHT for** prod **from 1 TO** N

ملاحظة :

الكلمات المسطّر تحتها تُمثّل الكلمات الواجب برمجتها أو العمليات الواجب إجرائها .

القسم الثالث من الفاتورة وهو الخاص بكل سلعة على حدة ، أي لكل رقم سلعة ، كمية . سعر السلعة ، السعر بدون الضريبة PHT (prix hors taxe) على الشكل التالي :

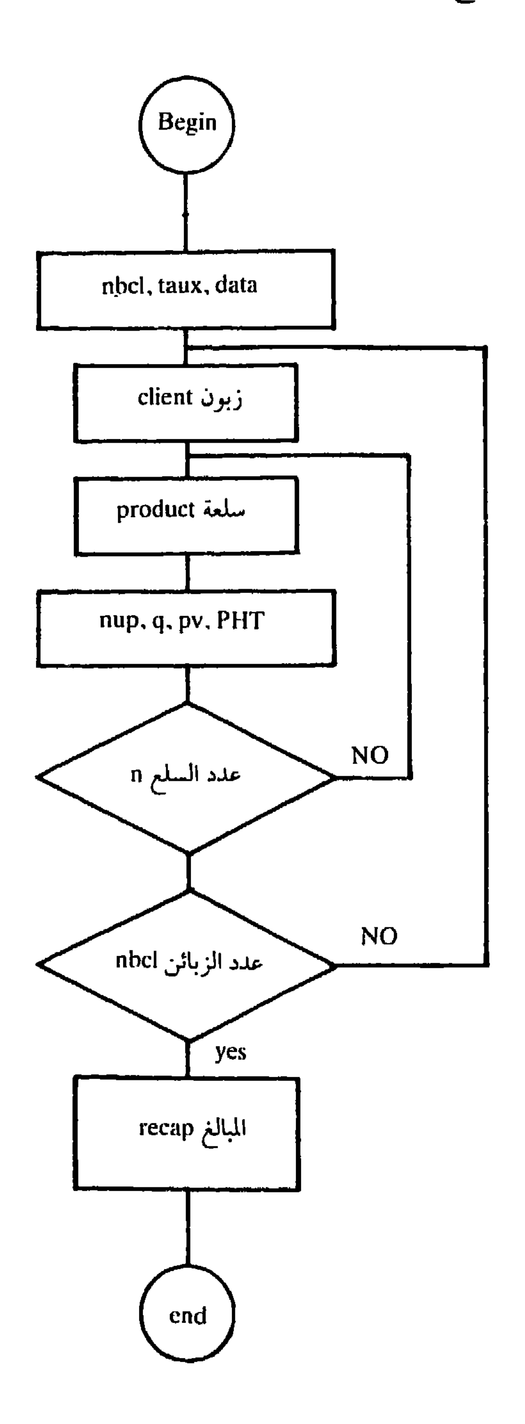
NUP, Q, PU = data ('numero du produit, quantité, prix unitaire: ')

PHT = Q × PV

هكذا فالأقسام الرئيسية من البرنامج يمكن أن تكون مرتبة بشكل منفصل على الشكل التالي :

النموذج الرئيسي	فاتورة (facture) لكل زبون من 1 إلى عدد الزبائن (nbcl)	السلع (products)	لكل سلعة من 1 إلى n (عدد السلع)

الخوارزم العام للبرنامج هو :



البرنامج بلغة باسكال :

```
PROGRAM FACTUR MUL;
    VAR NOM, VILLE, DATE : STRING  15;
        PRENOM : STRING [ 12 ] ;
        RUE: STRING [ 20 ] ;
        NUMR, NU?, Q, N, PROP: INTEGER;
        PV, PHT, THT, TAXE, TTC, TAUX: REAL;
        RES : TEXT;

    Begin REWRITE (RES, 'PRINTER : ');
        WRITE ('NOM, PRENOM, numero, rue ville du client:');
        READLN (NOM) ; READLN (PRENOM); READLN (NUMP);
        READLN (RUE); READLN (VILLE);
        WRITE ('DATE:') ; READLN (DATE);
        WRITE ('Nombre d'articles commandés:');

                                        { عدد السلع المطلوبة }
        READLN (N);
        WRITELN (RES);
        WRITELN (RES, »commande du', DATE);
        WRITELN (RES);
        WRITELN (RES, 'Numero du produit': 15, 'guantile': 10,
                    'PRIX UNIT': 10, 'PRIX HT': 10);
        THT: = 0;
        FOR PROD: = 1 TON DO
        BEGIN
        WRITE ('Numero de produit, quantité et prix unitaire:');
        READLN (NUP, Q, PU);
        PHT : = Q × PU;
        WRITELN (RES, NUP: 15, Q: 10, PU: 10: Z, PHT: 10: 2);
```

```
            THT : = THT + PHT
            END;
            TAXE : = THT + TAUX;
            TTC : = THT + TAXE;
            WRITELN (RES); WRITELN (RES, 'TOTAL hors taxe: ', THT
            10: 2);
            WRITELN (RES, 'T.V.A': tax: 10: 2);
WRITELN (RES, 'TOTAL TTC: ', TTC : 10: 2);
    CLOSE (RES, TOCK)
END.
```

هذا البرنامج يسمح بكتابة النتائج على الطابعة :

```
RES : TEXT;
RWRITE (RES, »PRINTER:');
```

10.4 ـ البحث في جدول من السمات

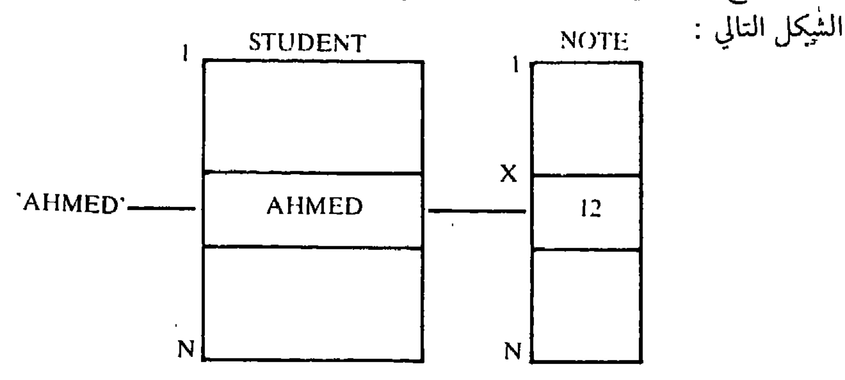
مسألة :

إكتب البرنامج الذي يسمح بالبحث عن النتيجة النهائية لكل طالب في إمتحان معين .

يجب أن يكون بتصرفنا جدولاً للتناسب بين الطلاب والنتائج STUDENT → NOTE ، هذا الجدول سيتم التصريح عنه بلغة باسكال على الشكل التالي :

```
STUDENT : array [ 1..N ] of packed array [ 1..N ] of char;
NOTE : array [ 1..N ] of integer;
```

موقع الاسم في الجدول الأول يُشير إلى نتيجة الطالب في الجدول الثاني ، كما نرى في الشكل التالي :



من هنا نحصل على المسودة الأولى للخوارزم الذي يبحث عن إسم مُعين بداخل جدول من الأسماء ، وعندما يجده يقوم بالبحث عن نتيجته من داخل جدول آخر .

```
1) program Research (input, output);
```

{ البحث عن نتيجة كل طالب }

```
Const N = ?; L = ?;
end = 'end';
Type chain = packed array [ 1..N ] of char;
Var STUDENT : array [ 1..N ] of chain;
    NOTE : array [ 1..N ] of integer;
    Nom : chain

begin
1- initialisation of student and note arrays
2- repeat
3- name request
4- comput the index x into NOT array
5- write NOT [ X ]
end.
```

- - N : يعادل عدد الطلاب .
- L : الطول الأكبر للإسم .

هكذا يحتوي جسم البرنامج على المسائل التالية :

- إعداد جداول الطلاب والنتائج .
- كرِّر .
- إطلب الإسم .
- إحسب المؤشر X في الجدول STUDENT المناسب للإسم .
- إكتب النتيجة NOTE [X] . .
- ما عدا الاسم الأخير 'END'
- النهاية .

2 ـ إعداد جداول أسماء الطلاب (STUDENT) ونتائجهم (NOTE) .

نفترض وجود N من أسطر المعطيات ، كلٍ منها يتألف من عدد صحيح ومن سلسلة رمزية .

```
procedure intialization
Var line: integer;
begin
    for line : 1 TON do
    read NOTE [ line ] and STUDENT [ line ] , GOTO
    next line
    end;
```

3 ـ الأمر : « إقرأ النتيجة NOTE [line] زللطالب STUDENT [line] ، وإذهب إلى السطر التالي » ، يترجم هذا الأمر على الشكل التالي :

```
(read NOTE [ line ] and STUDENT [ line ] . goto next line)
```

هذا الأمر يُكتب على الشكل التالي :

```
read (NOTE [ line ] );
 read STUDENT [ line ]
READ LN
```

4 ـ الأمر : « إقرأ STUDENT [line] » يؤدي إلى قراءة سلسلة غير كاملة ، أي إن الإجراء read nom [line] سيبدو على الشكل التالي :

```
procedure read chain (Var C: chain);
    var i, j : integer;
begin
    i : = 0;
while not eoln do begin i : = i + 1;
read (C [ i ]       ) end;
for j : = i + 1 To L do
    C [ J ] : = '   ';
readln
end;
```

سنختزل الحالة التي تكون فيها السلسلة طويلة ، أو عندما نقف في نهاية السجل .

5 ـ الأمر : repeat ... exclut يُترجم بواسطة ..While مسبوق بالحصول على العنصر الأول أو الاسم الأول للمعالجة .

ـ أطلب الاسم (name request)

يترجم :

While nom < > 'end' do begin

ـ إحسب مؤشر الاسم X في جدول الأسماء ، وإبحث عن النتيجة المناسبة لهذا الاسم : . NOTE [X]

ـ إطلب الاسم .

ـ النهاية .

11.5 ـ الفرز بالتبادل المتتالي

مسألة :

المطلوب فرز عناصر السلسلة T_n وعددها n حسب الترتيب التصاعدي ، أي الأصغر فالأكبر ، بشكل يصبح معه كل عنصر أصغر من جميع العناصر اللاحقة ، أي :

$$i > j \rightarrow T_i \geqslant T_j$$

الفكرة البسيطة المستعملة وتقوم على إتباع طريقة التتالي :

ـ إذا وضعنا في الموقع الأول من السلسلة المؤلفة من i من العناصر ، العنصر الأصغر ، فمعنى ذلك بأننا سنقوم بفرز i − 1 من العناصر .

ـ إذا تابعنا نفس الفكرة بالنسبة للعناصر i − 1 الباقية ، سيبقى معنا في النهاية عنصر واحد وبالتالي تكون السلسلة مفروزة بالكامل .

مثلاً :

T_i = 4 9 2 0 3

i = 1 2 9 4 0 3

0 9 4 2 3

i = 2 0 4 9 2 3

2 9 4 3

```
i = 3        4  9  3
             3  9  4
i = 4        4  9
```

السلسلة المفروزة : 0 2 3 4 9

```
program (input, output);
{ الفرز بالتبادل المتوالي }
const n = 5;                                { عدد العناصر المطلوب فرزها }
Var T : array [ 1..n ] of integer;          { السلسلة المطلوب فرزها }
    i, j, k, l : 1 .. n;
    aux : integer
begin
{ قراءة السلسلة بحالتها الأولى }
for k : = 1    to n do
    read ( T [ K ] );
  { الحلقة i : ضع العدد الأصغر الموجود في السلسلة T [ i..n ] في الموقع T [ i ] }
for i : = 1      To n – 1      do
     { الحلقة J : قارن العنصر T [ i ] بكل عنصر من السلسلة T [ i + 1 .. n ] }
for J : = i + 1 to n do
if T [ i ] > T [ J ] then begin
                                                { بداية التبادل }
aux : = T [ i ] ;
T [ i ] : = T [ J ] ;
T [ J ] : = aux
end;
                                                { اكتب السلسلة المرتبة }
for L : = 1 To n do write ('    ', T [ L ] );
writeln
end.
```

11.6

لنفترض الجدول A(m, n) الذي يحتوي على m سطر و n عامود . إكتب الخوارزم الذي يسمح بتبديل الأسطر i بالأعمدة j .

الحلّ :

لا نستطيع إجراء التبديل الشامل لكامل السطر i مع السطر j . يجب أن نقوم بعملية التبديل بشكل متتالي أي لكل عنصر من السطر i مع العنصر المناسب من السطر j . هذا ما يؤدي إلى تبديل ، ولجميع القِيم من k بين 1 و n ، مضمون الخلايا A(i, k) وA(j, k) ، للقيام بذلك يجب أن نستعمل خلية إضافية ثانوية تسمح بتخزين مضمون واحدة من هاتين الخليتين .

التبديل يتم على الشكل التالي :

$$X \leftarrow A(i, k)$$
$$A(i, k) \leftarrow A(j, k)$$
$$A(j, k) \leftarrow X$$

الخوارزم يبدو على الشكل التالي :

```
Var I, J, K, M, N : INTEGER;
    X : REAL;
    array A (30, 20) : real;
    read M, N
    read A
    read I, J
    For K from 1 TON repeat
                  X ← A (I, K)
                  A (I, K) ← A (J, K)
                  A J, K ) ← X
    Write A
```

مسألة :

لنفترض متعدِّد الجذور p(x) من الدرجة n ، والمُحدَّد بواسطة المُعاملات $a_{n+1} \ldots a_3$, a_2 , a_1 :

$$P(x) = a_1 x^n + a_2 x^{n-1} + \ldots + x_{n+1}$$

1 ـ فلنجمع على التوالي الآحاديات $(a_n x^x)$ (monôme) لتشكيل $(P(x_0))$

2 ـ باستطاعتنا أن نلاحظ إن :

$$P(x_0) = (((\ldots (a_1 x_0 + a_2) x_0 + a_3) x_0 + a_4) \ldots.) x_0 + a_{n+1}$$

(مخطط هورنر) .

فلنحسب $P(x_0)$ باستعمال هذه الملاحظة :

3 ـ فلتشكّل جدول القيم $P(x)$ بالنسبة لـ x المُتغيِّرة من b إلى c بخطوة Bh ليست بالضرورة قاسم $(c - b)$.

نستعمل الجدول COEF لتخزين مُعاملات متعدِّد الجذور . COEF (j) يحتوي إذاً على مُعامل الأحادية من الدرجة $n - j + 1$ و $P(a_n .02 j + 1)$.

أ ـ السؤال الأول

يجب أن نحسب القوى المتتالية لـ x_0 .

نحصل على $(x_0)^{j-1}$ من خلال $(x_0)^j$ ، وذلك بضرب $(x_0)^j$ بـ x_0 . سنستعمل أسماء المتحولات التالية :

ـ x_0 : لتخزين قيمة x_0 .

ـ COEF : جدول يسمح بتخزين مُعاملات متعدِّد الجذور .

ـ N : درجة متعدِّد الجذور .

ـ P : لتخزين قيمة المجموعات الجزئية للآحاديات $(u_n x^n)$ ، والقيمة $P(x_0)$.

ـ j : عداد التكرار .

ـ PUIS : لتخزين القوى المتتالية (الأس) لـ x_0 .

سنفترض ، إضافة لذلك ، إن متعدِّد الجذور هو بدرجة أصغر أو تساوي 20 ، وإذا لم يكن كذلك ، يجب التصريح عن الجدول COEF بأبعاد أكبر .

الخوارزم :

```
Var J, N : INTEGER ; X0, P, PUIS: REAL;
    Tab COEF (20): REAL;
read N, X0
read COEF
P ← 0
```

(إعداد المجموعات الجزئية وتصغيرها)

```
PUIS ← 1 ((x¹) x0 «gx 1 لـ إعداد القوى المتتالية )
for J from N + 1 TO 1 by step − 1 repeat
P ← (PUIS ✱ COEF (J) + p
    PUIS ← PUIS ✱ x0 ( X0 حسبان القوة التالية لـ )
write P
```

ب ـ السؤال الثاني :

الطريقة المعروضة ، تقوم على حسبان متتالي لقيمة متعدِّد الجذور P_i على الشكل التالي : $p_i = p_{i-1}\, x_0 + a_i$. القيمة $p(x_0)$ هي القيمة الأخيرة التي نحصل عليها ، لنفترضها p_{n+1} .

جميع القيم p_i ستكون على التوالي مخزَّنة في نفس الذاكرة p .

الخوارزم :

```
Var J, N : INTEGER ; x0, p: REAL ; tab COEF (20): REAL
read N, x0
read COEF
P ← COEF (1)                                  ( إعداد P إلى a1 )
for y from 2 TO N + repeat
P ← P ✱ x0 + COEF (J)
write P
```

نلاحظ إن هذه الطريقة تستعمل أقل بمرتين عمليات الضرب من السابقة : فهي عملياً أسرع بمرتين . مدة العمليات الأخرى هي عملياً لا تؤخذ بالاعتبار .

السؤال الثالث :

سنستعمل الجدول VAL ، حيث سنُخزِّن في الموقع رقم i ، القيمة رقم i لـ x (أو x_i) من الفسحة [a, b] .

الجدول TP سيسمح بتخزين القيمة المناسبة p(x) في الموقع رقم i .

نحسب القِيم p(x) لـ x = a ، x = a + h ، x = a + 2h . . . القيمة الأولى الأعلى

من b من هذه القيم . سيتم إستبدالها بـ x = b . نكرِّر حسبان p(x) لجميع هذه القيم ، ونُوقف التكرار بعد حسبان p(b) . نستعمل المُتحولة المنطقية لمعرفة إن x قد بلغت القيمة b .

مخطط الخوارزم :

- قراءة درجة متعدِّد الجذور
- قراءة مُعاملات متعدِّد الجذور
- قراءة a . قراءة b . قراءة الخطوة h .
- إعداد عداد الأسطر j وإعطاؤه القيمة 1 .
- نضع الحدَّ الأدنى للفسحة في الخلية V من الذاكرة .
- إعداد المتخولة المنطقية وإعطاؤها القيمة FALSE .
- كرِّر إذا كانت v ≥ b فإذاً ستصبح المتحولة المنطقية « حقيقة » (true) ، وإذا كانت v < b نعطي القيمة b إلى v .
  حسبان p(v) .
  وضع النتيجة الحاصلة في الموقع j من الجدول TP .
  وضع V في السطر j من الجدول VAL .
  المرور إلى القيمة التالية من الفسحة (وضعها في V) .
  حتى تصبح المتحولة المنطقية معادلة لـ «true»
  إكتب VAL
  إكتب TP .

لحسبان p(x) باستطاعتنا أن نستعمل إحدى الطرق السابقة (هنا سنستعمل الطريقة الثانية) .

نفترض بأننا نأخذ 30 قيمة على الأكثر في الفسحة [a, b] ، أي إن

$$h > \frac{(b-a)}{30}$$

الخوارزم

Var J, N, K : INTEGER ; A, B, H, P : REAL ; STOP : BOOLEAN; tabCOEF (20), VAL (30), TP(30) : REAL;

read N

read COEF

```
read A, B
read H
J ← 1
V ← A
STOP ← FALSE
repeat    if V ≥ B then STOP ← TRUE
          if V > B then V ← B
```

أ (حسبان قيمة متعدِّد الجذور بالنسبة لـ V)

```
          PP ← COEF (1)
          for K from 1 TON repeat
          PP ← PP * V + COEF (K + 1)
          VAL (j) ← PP * V + COEF (K + 1)
          if STOP = FALSE then V ← V + H
          J ← J + 1
UNTIL STOP = TRUE
 for k from 1 to j repeat
write VAL (K) , TP (k)
```

ملاحظة :

إستعمال المتحولات PP وV ، يبدو وكأنه بدون معنى . ولكن إذا قمنا باستبدالهما بـ P(j) وVAL (J) في جميع الأوامر ، نخسر كثيراً من الوقت عند التنفيذ لحسبان العناوين المناسبة في كل مرَّة (عنوان العنصر رقم j من الجدول = عنوان بداية الجدول – 1 + (j) * عدد كلمات العنصر) .

مسألة : الرسم البياني (graphe)

لنفترض رسم بياني موجه ، ويتمثل بواسطة عدد يساوي N من القمم ، والأقواس التي تربط بعض هذه القيم .

لكل قمة من القمم ، باستطاعتنا تعريف لائحة القمم k بحيث يكون كل (k, j) عبارة عن قوس من الرسم البياني (k هي المركز وj هي طرف القوس) . هذه اللائحة تدعى لائحة القمم السابقة للقمة j .

نفس الشيء، لكل قمة j ، يُمكن أن نعرِّف القمم k . بحيث إن كل (j, k) هو عبارة عن قوس من الرسم البياني (j هي المركز و k هي طرف القوس) . هذه اللائحة تدعى لائحة القمم اللاحقة للقمة j .

إكتب البرنامج الذي ، من خلال N لائحة من القمم السابقة يصنع N لائحة من القمم اللاحقة .

تكوَّد القمم بواسطة أعداد صحيحة من 1 إلى N .

نفترض إن لوائح القمم السابقة لكل قمة هي مخزنة في جدول من الأعداد الصحيحة يُدعى ANT ، هذا الجدول هو ببعدين ، ويكون على الشكل التالي :
- على السطر j ، نجد لائحة القمم السابقة للقمة j .
- إذا كانت القمة j تتمتع بعدد K من القمم السابقة . سنحصل على :

$$ANT (j, k + 1) = 0$$

المستوى الأول للتحليل

نستعمل جدولاً من الأعداد الصحيحة ببعدين لتخزين لوائح القمم اللاحقة لكل قمة يدعى PUI ، وذلك حسب نفس طريقة تشكيل الجدول ANT بالنسبة للقمم السابقة .

مخطط التحليل

- لكل قمة نقرأ لائحة القمم السابقة لها .
- لكل قمة :

نبحث عن لائحة القمم اللاحقة ، وتخزين هذه اللائحة في الجدول SUI وذلك على السطر المناسب .

- لكل قمة :

نكتب لائحة القمم اللاحقة .

الخوارزم :

```
Var J, K, N : array ANT (100, 200) ; integer;
    array SUI (100, 20) : integer;
read N
For J = 1     TO N repeat
```

```
    repeat k ← k + 1
          read ANT (J, K)
until ANT (J, K) = 0
for j = 1 TO N repeat
```

نبحث عن القمم اللاحقة للقمة j ونقوم بتخزينها في الجدول SUI على السطر j

```
for j = 1 TO N repeat
k ← 1
while SUI (J, K) = 0 repeat write SUI (j, k)
                                        k ← k + 1
GOTO new line
end.
```

ملاحظة :

من الممكن أن نشير إلى الفرق بين اللائحة من نوع «repeat .. until» عند القراءة والحلقة من نوع while عند الكتابة أو الإخراج .

تحليل بالمستوى رقم 2

نقوم ببناء وإنشاء إجراء (procedure) ، يقوم ، ومن خلال قمة معينة ، بإنشاء لائحة القمم اللاحقة . لنفترض إن EXTERN هو إسم هذا الإجراء .

ويتمتع بالمتغيرات الوسيطية التالية :

ـ المتغيرات المُعطاة :

ـ ANT : جدول القمم السابقة لكل قمة .

ـ N : عدد الأسطر من الجدول ANT .

ـ X : كود القمة التي نبحث عن القمم اللاحقة لها .

ـ متغيرات الناتج .

ـ LIST : لائحة القمم اللاحقة للقمة X .

لكي تكون القمة Y هي طرف لقوس بمركز X ، يجب وفقط يكفي أن تنتمي X إلى لائحة القمم السابقة لـ Y .

لكي نقوم بإنشاء لائحة القمم اللاحقة للقمة X ، سنقوم باختيار إذا كانت X.

موجودة في لائحة القمم السابقة للقمة K ، وذلك لكل قمة K ، فإذا كانت كذلك ، نُضيف K إلى لائحة القمم اللاحقة للقمة X .

```
procédure EXTREM (ANT, N, X, LISTE)
parameters : N, X : entier;
    array ANT (100, 20): integer;
results : ARRAY LISTE (20) : integer;
    Var K, L : integer;
L ← 0
for K from 1 TO N repeat
```

نبحث عما إذا كانت القمة X موجودة على لائحة القمم السابقة للقمة K

```
if X 6 ANT then L ← L + 1
                    LIST (L) ← k
L ← L + 1
LISTE (L) ← 0
return
```

التحليل بالمستوى الثالث :

نقوم بإنشاء الإجراء CHERCHE ، الداخلي ضمن الإجراء EXTREM ، والذي يبحث عما إذا كانت القمة X_1 موجودة في لائحة القمم السابقة للقمة X_2 .

متغيرات الإجراء CHERCHE هي :

- متغيرات داخلة :
- ANT ، جدول القمم السابقة .
- X_1 ، القمة التي نبحث عنها في لائحة القمم السابقة لـ X_2 .
- X_2 ،
- متغيرات ناتجة أو النتائج .
- VERIF : متغيرة من نوع منطقي ، هو ان المتحولة ستحتوي على القيمة true إذا كانت X_1 موجودة في لائحة القمم السابقة للقمة X_2 ، وFALSE في الحالة الأخرى .

وكما إن الإجراء procedure لا يتمتع إلا بمتغير وسيطي ناتج ، فبإمكاننا التعريف عن الدالة .

```
fonction CHERCHE (ANT, X1, X2) : BOOLEAN;
data Parameters: x1, x2 : integer Tab ANT (100, 20) : integer
var J : integer
j ← 1
while ANT (X2, j) = 0 and ANT (X2, j) = X1 repeat
                                    j ← j + 1
if ANT (X2, j) = X1 then CHERCHE ← TRUE
                         else CHERCHE ← FALSE
return
```

بإمكاننا الآن كتابة الاجراء EXTREM الذي يُنادي الدالة CHERCHE .

```
procédure EXTREM (ANT, N, X, LISTE)
data parameters : N, X: integer; ARRAY ANT (100, 20): integer;
result parameters : ARRAY LISTE (20) : integer;
    Var K, L : integer
    L ← 0
    for K from 1 TO N repeat
        if CHERCHE (X, K) = TRUE then L ← L + L
                                    LISTE (L) ← K
l ← L + 1
LISTE (L) ← 0
return
```

بإمكاننا الآن كتابة البرنامج الأساسي حيث التحليل يناسب المستوى واحد .
نستعمل الجدول SV ببعد واحد (متجه vector) لتخزين لائحة القمم اللاحقة للقمة j
عند دعوة الاجراء EXTREM ، هذا الجدول هو أيضاً مرتب في السطر j للجدول SUI .
فهو إذاً قابل للاستعمال للقمم اللاحقة للقمة j + 1 .

البرنامج الأساسي :

```
Var Y, K, N, ARRAY SU (20), ANT (100, 20), SUI (100, 20) : integer;
    read N
```

```
for J from 1 TO N repeat
    K ← 0
    repeat K ← K + 1
        read ANT (J, K)
        until ANT (J, K) = 0
for j = 1 TO N repeat
call EXTREM (ANT, N, J, SU)
K ← 0
repeat k ← k + 1
    SUI (J, K) ← SU (K)
    until SU (k) = 0
to j from 1 TO N repeat
    K ← 1
     while SUJ (j, k) = repeat
write SUI (j, k)
k ← k + 1

end.
```

ملاحظة :

ـ إذا كانت لغة البرمجة تسمح باقتسام المتحولات بين البرنامج المركزي والاجراءات ، فمن الممكن الافتراض إن المتحولات ANT وSV هي مبلوغة مباشرة بواسطة الاجراءات ، مما يؤدي إلى تفادي التصريح عنها بشكل جلي كمتغيرات في الاجراء .

ـ لقد إفترضنا إن عدد القمم التي تؤلف الرسم البياني هو أقل أو يساوي 100 ، وإن لكل قمة 20 قمة سابقة و20 قمة لاحقة على الأكثر .

خاتمة

يعتبر الخوارزم من أهم المراحل في علم البرمجة ، ووضعه يتطلب إلماماً كبيراً بموضوع المسألة وذلك بهدف إختيار الطريقة المناسبة للحلّ . ويعتمد المُحلّل عند وضعه للخوارزم على الطريقة الرياضية المناسبة لبلوغه الهدف بأقل قدر ممكن من الأخطاء ، وبالسرعة الممكنة الأكبر لتنفيذ البرنامج ، كما ويجب على المُحلّل أن يأخذ حجم الذاكرة بالحسبان عند وضعه للخوارزم وذلك بهدف تخفيض الحجم المشغول قدر الإمكان ، والتوفير في إمكانيات المكنة ومقدراتها .

ويجب على المُحلل والمبرمج في آن أن يتعرّف على طرق بناء وإنشاء المعطيات وإختيار التركيبة أو البناء الملائم لتسجيل معطيات وذلك بهدف تسريع البلوغ إليها وتخفيض حجم الذاكرة الداخلية والخارجية الذي قد ينشغله البرنامج والمعطيات . لهذا كله يُعتبر التعبير التالي : خوارزم + تركيب المعطيات يعادل البرنامج من التعابير الدقيقة والفعالة في عالم البرمجة .

وفي هذا الكتاب توخينا إعطاء لمحة شاملة عن كيفية إنشاء الخوارزميات ، وبناء المعطيات للحصول على البرامج المختلفة . كما اعتمدنا لغة باسكال كلغة تصلح للبرمجة الانشائية لصياغة الأمثلة والمسائل المختلفة .

المراجع

— Algorithmique. construction preuve et évaluation des programmes.
par Pierre BERLIOUX, PHILIPPE BIZARD DUNOD 1983 - PARIS.

— Cours dalgorithmique. PAR DR. ABOUL HASSAN HUSSEINI.
univ. de libanaise, fac. de genie - 3- BYROUTH.

— Exercises commentés d'analyse et de programmation par : J- P LAURENT,
J. AYEL DUNOD 1986 PARIS.

— Initiation à l'algorithmique et aux structures de données. programmation
structurée et structures du données par : J. COURTIN, I. KOWARSK;
DUNOD 1987- PARIS.

— Programmation TOM 1, 2. du problème à l'algorithme et de l'algorithme au
programme
Amedée DUCRIN
DUNOD 1985 . PARIS.

— FORTRAN structuré et methodes numeriques.
S. FAROULT, D. SIMON
DUNOD. 1986- PARIS.

— Raisonner pour programmer.
Anna GRAM.
DUNOD 1986 PARIS.

— Initiation à d'Algorithmique
C. et P. Richard

edition BELIN 1986 PARIS.

— DATA structure
from algorithme to program
N- WIRTH 1978 MACGRAWHILL

— WIRTH. N. PROGRAMING language pascal 1971.

— systematic programing - An introduction
prentice Hall.1973
N- WIRTH

— Algorithm + data structure .004 programs
N- WIRTH.
prentice Hall 1976.

— IBM 300/370 PASCAL 8000 for OS/VS
University of TOKYO. Rewriten for australian energy commission 1978.

— KRUCHTEN P.
Langage de programmation pascal. ey rolles 1979.

— pascal par l'exemple
J.A. FERNANDEZ
Eyrolles 1980.

— Introduction au pascal
BEUX. P
SYBEX 1980.

فهرست






المؤسسة الجامعية للدراسات والنشر والتوزيع